Regular Paper

C4ISR Architectures: II. A Structured Analysis Approach for Architecture Design

Lee W. Wagenhals, Insub Shin, Daesik Kim, and Alexander H. Levis*

System Architectures Laboratory, C3I Center, MSN 4D2, *George Mason University, Fairfax,* VA 22030

Received April 25, 2000; accepted July 15, 2000

ABSTRACT

A Structured Analysis based process for developing C4ISR architectures is presented. The process demonstrates the feasibility of developing architecture descriptions that conform to the C4ISR Architecture Framework based on the Structured Analysis paradigm that underlies the concepts and definitions in the Framework. Furthermore, the process incorporates the derivation of an executable model that can reveal the logical, behavioral, and performance characteristics of the architecture. The complete process is illustrated through an example involving the insertion of a new technology in a large legacy system. © 2000 John Wiley & Sons, Inc. Syst Eng 3: 248–287, 2000

1. INTRODUCTION

This is the second of three articles on Command, Control, Communications, Computers, Intelligence, Surveillance and Reconnaissance (C4ISR) Architectures. In the first article, Part I [Levis and Wagenhals, 2000], we described the need for architectures, the role of the architect in producing them, and described an approach to architecting information systems that was based on Structured Analysis. We also described briefly the C4ISR Architecture Framework, Version 2 [C4ISR, 1997] and its products noting that the Framework purposely does not present or mandate a process for producing those products. Finally, we postulated that two alternative, distinct approaches may be taken to develop architectures and the Framework products. One approach is based on Structured Analysis concepts, and the other is based on Object-Oriented ones [Bienvenu, Shin, and Levis, 2000]. Both approaches produce products that contain a great deal of information. Unfortunately, these products are static representations of a dynamic system and therefore are less than ideal for illuminating behavior and performance characteristics.

We have developed procedures for deriving, from the information contained in the static architectural products, an executable model that can be used to

*Author to whom all correspondence should be addressed.
Grant sponsor: Office of Naval Research; grant number: N00014-00-1-0267
Grant sponsor: Air Force Office for Scientific Research; grant number F49620-95-0134
Systems Engineering, Vol. 3, No. 4, 2000

Table I. Inputs to the Process Collected in Stage 0

(AV1)	Purpose, Viewpoint (Problem Definition)
D1	Operational Concept Narrative
D2	Universal Joint Task List (UJTL)
D3	Current DOD Organization charts/ Joint, Services, Agencies
D4	Description of Organizational Relationships
D5	Textual description of Doctrine, Tactics and Operational Procedures
D6	List of Operational Information Elements
D7	Definitions of States and Events
D8	Description of System Functions
D9	Communication Systems Description
D10	Performance Attributes of Systems
D11	Migration Plans for Systems
D12	Description of Systems

demonstrate rigorously the behavior and performance characteristics of the architecture. Furthermore, the executable model can be used to verify the architecture design. Any changes that must be made in the executable model to get it to exhibit the desired behavior must then be reflected in changes to the static representations.

In this article, we present an approach for developing the C4ISR Framework products based on the traditional Structured Analysis approach. We illustrate the approach with a commercial case study in which a new technology is introduced for expanding the capability of the "pay at the pump" systems available at most gasoline stations. This is an example of information technology insertion in a large legacy system.

The remainder of this article is divided into six sections. Section 2 presents a process for developing a C4ISR architecture using the Structured Analysis approach and then creating the Framework products from that effort. An activity model is used to present the process. Section 3 describes the case study. In Section 4, we show how to convert the elements of the architecture to the discrete event dynamic model, a Colored Petri Net. In Section 5, we use the example of Section 3 to show how the executable model can be used to analyze the logical and behavioral characteristics of the architecture. Section 6 describes how the executable model can be extended to allow the architect to extract performance characteristics that can be discussed with the user or customer of the systems that will be built in conformance with the architecture. Section 7 presents overall conclusions.

2. THE STRUCTURED ANALYSIS APPROACH

The starting point for defining this process is developing an activity process model. Both IDEF0[1] and Data Flow Diagrams have been used. In both models, the context diagram contains a set of documents and information as inputs or terminators and the set of Framework products as the output or terminators of the process. The process has been divided into five steps or stages. Each stage generates at least one or more of the Framework products. This allows for continuous review and evaluation of the architecture design.

The first step in any architectural effort involves the collection of domain information. This is designated as Stage 0 in the proposed process. In the C4ISR context, 12 types of documents and information have been identified as candidates for this effort and they are listed in Table I. Once they are gathered, they form the input to the process and are represented as Terminators in the Data Flow Diagram description of the process.

This gathering of domain information is analogous to an architect eliciting requirements and desires from a client who wants the architect to design a new home. In the C4ISR context, this requirements elicitation process can be a little more formal. The users (the operators) and the customers (the acquisition executives) have a wealth of information about how the DoD does business, including formal documents that define doctrine and tactics, formal organizational structures, and descriptions and specifications of existing

[1]IDEF0 stands for Integrated Computer Aided Manufacturing (ICAM) Definition Language 0; it is documented in the Federal Information Processing Standard (FIPS) #183.

systems that may be incorporated in the design of the architecture.

The list of source documents in Table I represents typical classes of information. As was described in Levis and Wagenhals [2000], the architecting process must start with a clear purpose and viewpoint and an operational concept must be provided. Sources for these items may include mission needs analyses and operational requirements documents (ORDs) as well as discussions with operators of the systems that are similar to the ones that will be defined in the architecture. These documents and elicitations are listed as Purpose and Viewpoint, part of the All Views Overview and Summary Information (AV-1) product and D1, Operational Concept Narrative, in Table I.

DoD has published the Universal Joint Task List (UJTL) that appears as D2. This list is a high level functional decomposition of standard tasks and functions that are performed by DoD organizations during military operations. DoD and its components also have standardized organizational structures and relationships. They are described in standard command relationship charts and standing operational plans and operational orders. These also are important references to the architect and are listed as D3 and D4. As the architect delves more deeply into the background domain information, he/she can make use of DoD documents that describe doctrine and tactics, techniques, and procedures (D5). In some cases, the architect may be able to access studies about systems that reveal typical operational or system information elements. Many previous architectural efforts contain this type of information. Furthermore, documents describing standard message types also may be useful. These items are listed as D6, List of Operational Information Elements.

As the architect elicits material, users of the systems often describe their vision of the architecture in terms of events that the system must react to and various high level states that the system will be in. Examples include readiness states and the events or conditions that can occur that should cause the system to change states. These descriptions, listed as D7, are important to the architectural design and may offer high level descriptions of desired behavior.

As the architect creates the system architecture views, information about potential systems (D12) and their functions (D8) will be useful. A variety of documents provide technical descriptions of current and future systems that can serve this purpose and provide performance parameters of those systems (D10). In addition to the systems, descriptions of existing and planned communications networks and links also are available to the architect (D9). Finally, the architect may refer to documents that describe the planned evolution or migration of the system. DoD planning guidance and defense plans are good sources for this information (D12).

The five stages of the process are shown in the Data Flow Diagram of Figure 1. The diagram also shows the input terminators as described in Table I and the output terminators which are the C4ISR Architecture Framework Products. A list of products is given in the Appendix; for a description, see C4ISR [1997] or Levis and Wagenhals [2000].

Stage 1 is shown in Figure 2. While this is a very simple diagram, it represents a critical step in the architecting process because it is the operational concept that guides the remaining stages of the process; thus it is passed to the second stage as indicated by the "2" in the oval. The OV-1 product, the High Level Operational Concept Graphic, is produced in this stage.

Stage 2 is shown in Figure 3. It has four terminators that provide inputs and one terminator that is the output of the stage, the Command Relationship Chart (OV-4). In this stage, the architect uses the Joint Universal Task List and the Operational Concept (as shown by the "2" in the oval) to determine the functions that need to be performed to carry out the operational concept and organizes them in a functional decomposition. The architect also uses the operational concept, the list of potential organizations, and the organizational relationships to determine which organizations to include in the architecture and the command relationships that will exist between those organizations. These command relationship are documented in the Command Relationship Chart, OV-4. These organizations have assets that are the systems that will support the activities. The traditional or logical grouping of the organizations and their assets are used to define operational elements and aggregate them into the operational nodes that will be depicted in the operational node connectivity diagram.

In Stage 3, the architect performs the bulk of the structured analysis process that was described in Levis and Wagenhals [2000]. As shown in Figure 4, the architect is engaged in three major efforts. First, the functional architecture, composed of the activity model, the logical data model, and the rule model are created based on the functional decomposition. The desired behavior of the architecture is captured in the state transitions diagram. The concordance process ensures consistency and completeness of all of these products. Second, the architect creates the initial physical architecture composed of system nodes that contain systems, system components, and system elements. Consistency and balance is maintained between the physical and functional architecture views through two allocation processes, the third main activity of Stage 3. In the allocation process, the architect evaluates the assets possessed to

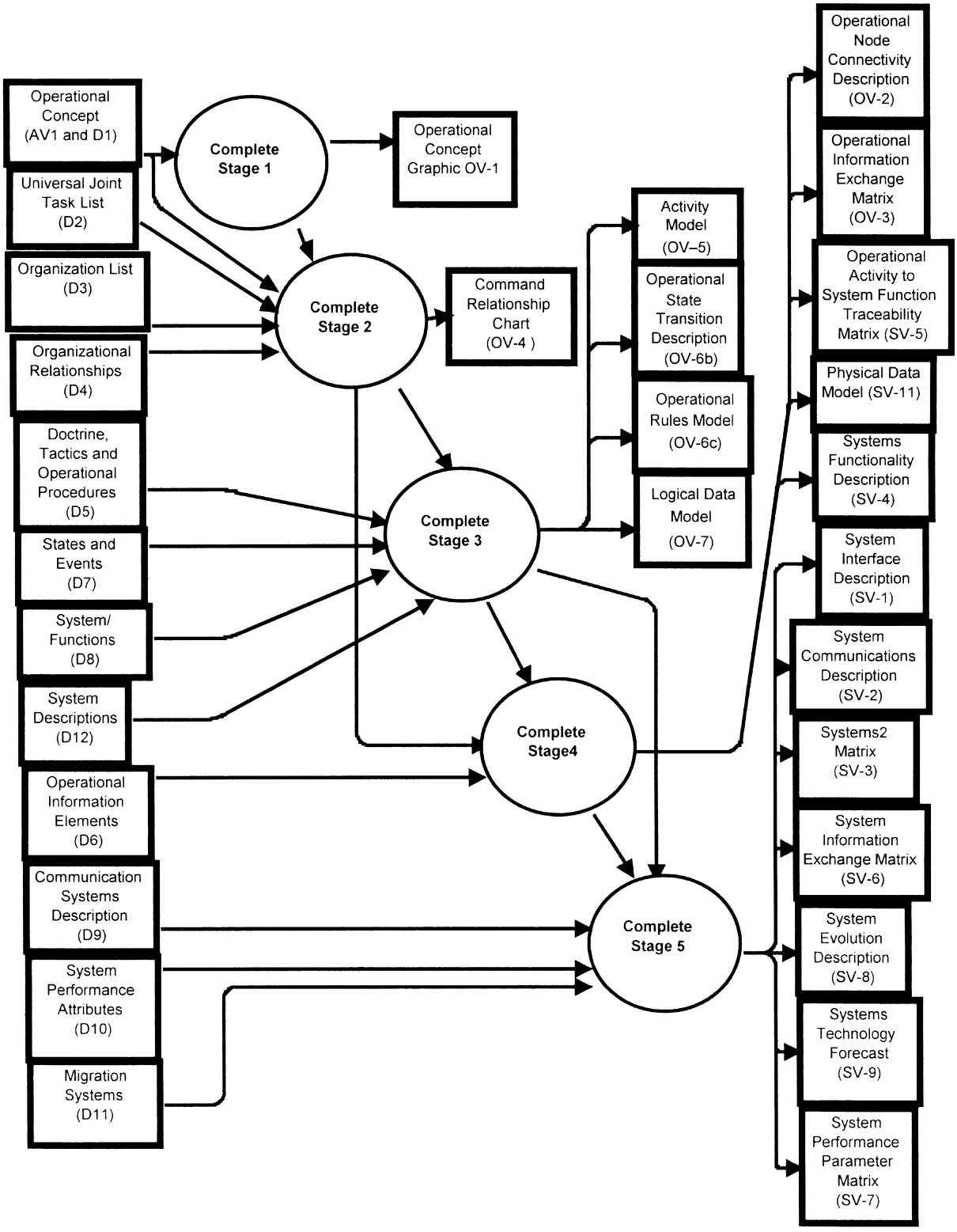


Figure 1. Data Flow Diagram of the five-stage process.

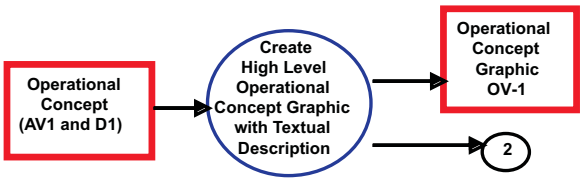


Figure 2. Process model of Stage 1: develop the operational concept.

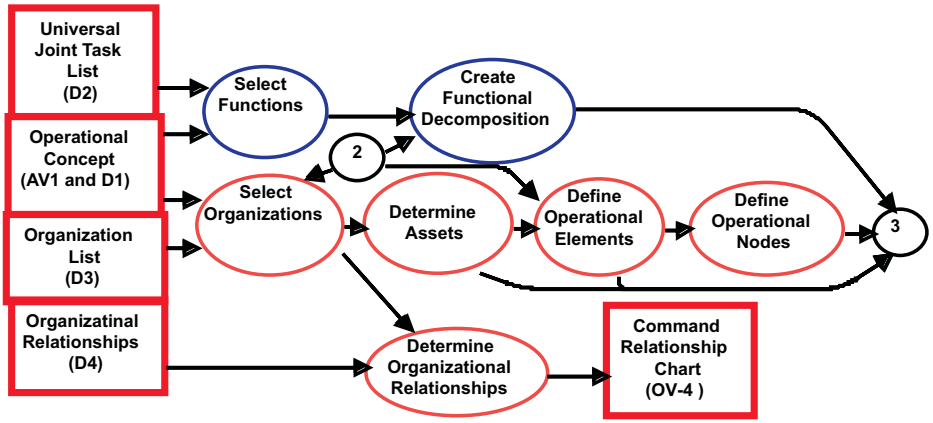


Figure 3. Process model of Stage 2.

be associated with by the operational elements. These assets are systems that perform system functions. The architect decides which system functions should perform the operational activities. This in turn allocates the activities in the activity model to those operational elements that own these systems. The activities are also allocated to the functions those systems perform. This is the key step that marries the operational and system architecture views together. All of the activities in Stage 3 are highly coupled and an iterative process is used during this stage.

In Stage 4, shown in Figure 5, the architect creates the remaining Operational Architecture View products using the information and models created in Stage 3. Key parts of the analysis needed to create the System Architecture View products are also done.

The Logical Data Model and the Needlines[2] define the Operational Information Elements. The allocation process assigned activities to operational elements and nodes. The activity model contains the information flows between activities and, by the allocation process, between operational nodes. Thus, the Operational Node Connectivity Description (OV-2) and the Operational Information Exchange Matrix (OV-3) can be extracted from the combination of activity model and allocation. The allocation of activities to system functions is documented in the Operational Activity to System Function Traceability Matrix (SV-5). Using this allocation and the activity model, the architect creates the System Functionality Description (SV-4), which is a process model that uses the system functions as the processes or transformations. SV-4 and the Logical Data Model (OV-7) can be used to create the Physical Data Model, SV-11.

Stage 5 is dedicated to completing the System Architecture View Products. As shown in Figure 6, it requires inputs from both Stages 3 and 4. The System Information Elements are the physical manifestation of the Operational Information Elements that were defined in Stage 4. They can be defined from a combination of the Physical Data Model (SV-11) and the System Activity Description (Data Flow Diagram of system functions). The allocation and initial physical architecture allows the definition of the system nodes that the system information elements flow between. Once the System Information Elements have been specified, the System Information Exchange Matrix (SV-6) can be created. The characteristics of the System Information

[2]Needlines are defined in C4ISR [1997] as requirements that are the logical expression of the need to transfer information among nodes.

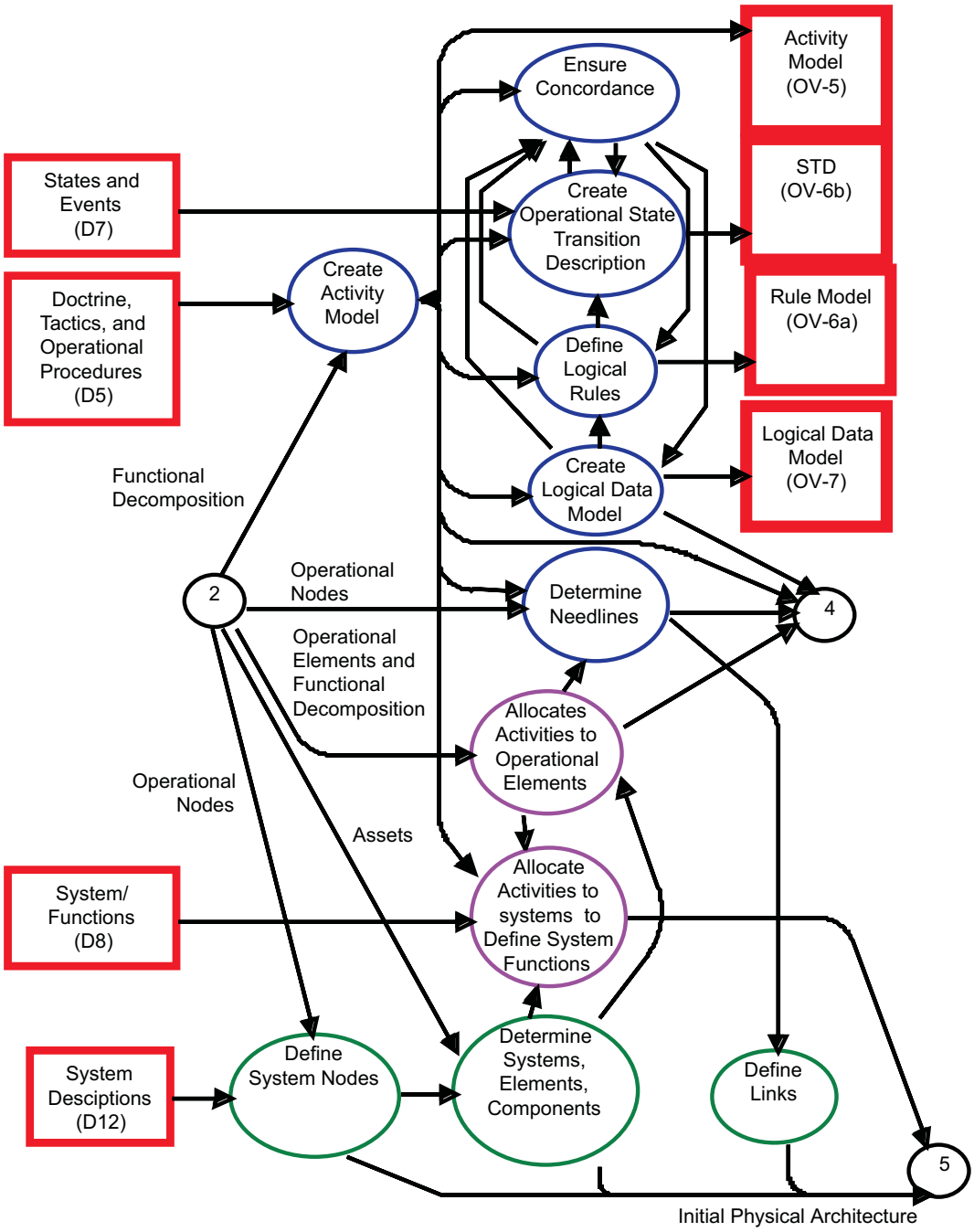


Figure 4. Process model of Stage 3.

Elements and Exchanges are used to guide the selection of the Communications System including the LANs and WANs that will be included in the architecture. The architect uses the Systems Communications Description (SV-4) that was obtained in Stage 0. Once the LANs and WANs have been identified, the architect can create the System Interface Description (SV-1).

The System Interface Description (SV-1) describes the physical manifestation of the Operational Node Connectivity Description (OV-2). It is based on the Initial Physical Architecture that was developed in Stage 3. The Systems Communications Description (SV-2) can be created using the LAN and WAN selections. It should also be coordinated with the System Interface Description because it is a more detailed view

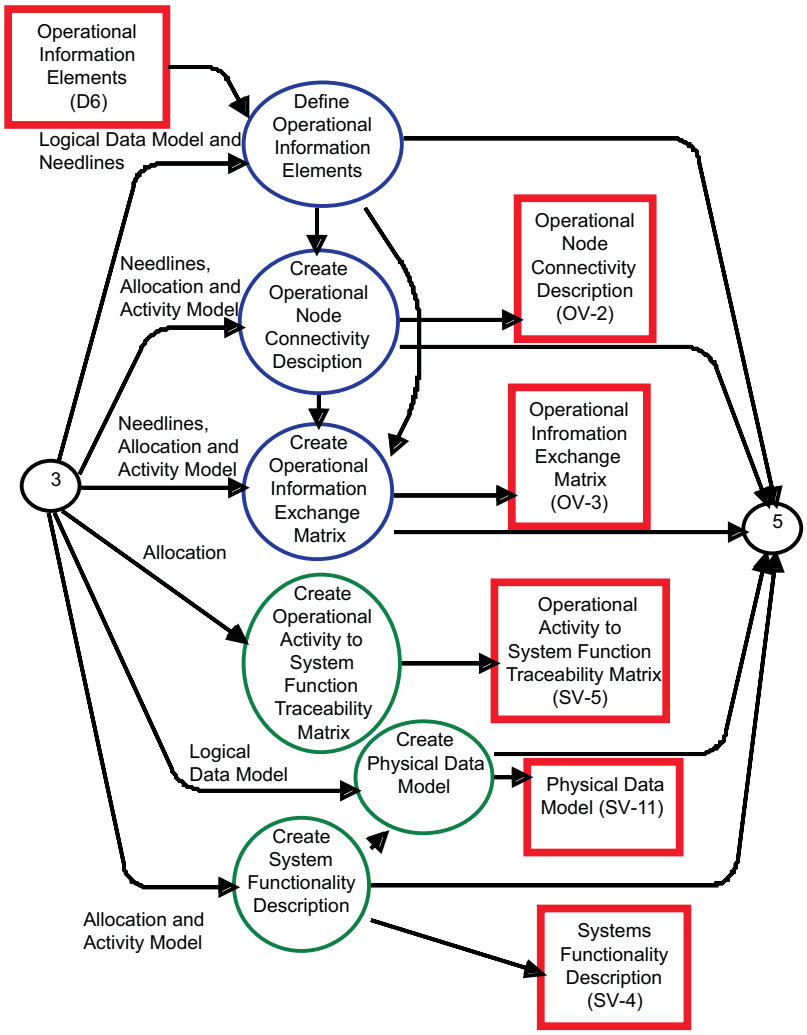


Figure 5. Data Flow Diagram of Stage 4.

of the Systems Architecture. The LANs and WANs define the interfaces between the systems, system elements, and system components. Once the interfaces have been defined, the System[2] Matrix (SV-3) can be created. Finally, the architect can create the System Evolution Description (SV-8), System Performance Parameter Matrix (SV-7), and the System Technology Forecast (SV-9) using the Initial Physical Architecture and the Migration Systems (D11) and System Performance Attributes (D10) information that were developed in Stage 0. Clearly, there is a great deal of redundancy in the System Architecture View. The architect must maintain consistency (concordance) between these products.

3. AN EXAMPLE OF THE PROCESS

The illustrative example is based on a relatively new product developed by the Mobil Corporation called the Mobil SpeedPass. However, this is not an accurate description of the system—it has been created for the express purpose of illustrating the architecture design process, especially the case where a new information technology providing a new capability is grafted on existing large legacy information systems. Consequently, it will be assumed that some oil company (OilCo) has implemented a new system called FastPass. This example has been chosen in lieu of a DoD example because of its familiarity to a large audience.[3] We start with a description of the goal of the system and proceed through the five stages presented in Section 2.

We assume that all of the initial research of Stage 0 has been completed and the input documents have been collected. Part of this process is the gathering of information about the operational concept. In this example,

[3]The example has been used as a one-day classroom exercise by over 300 students working in teams of four.

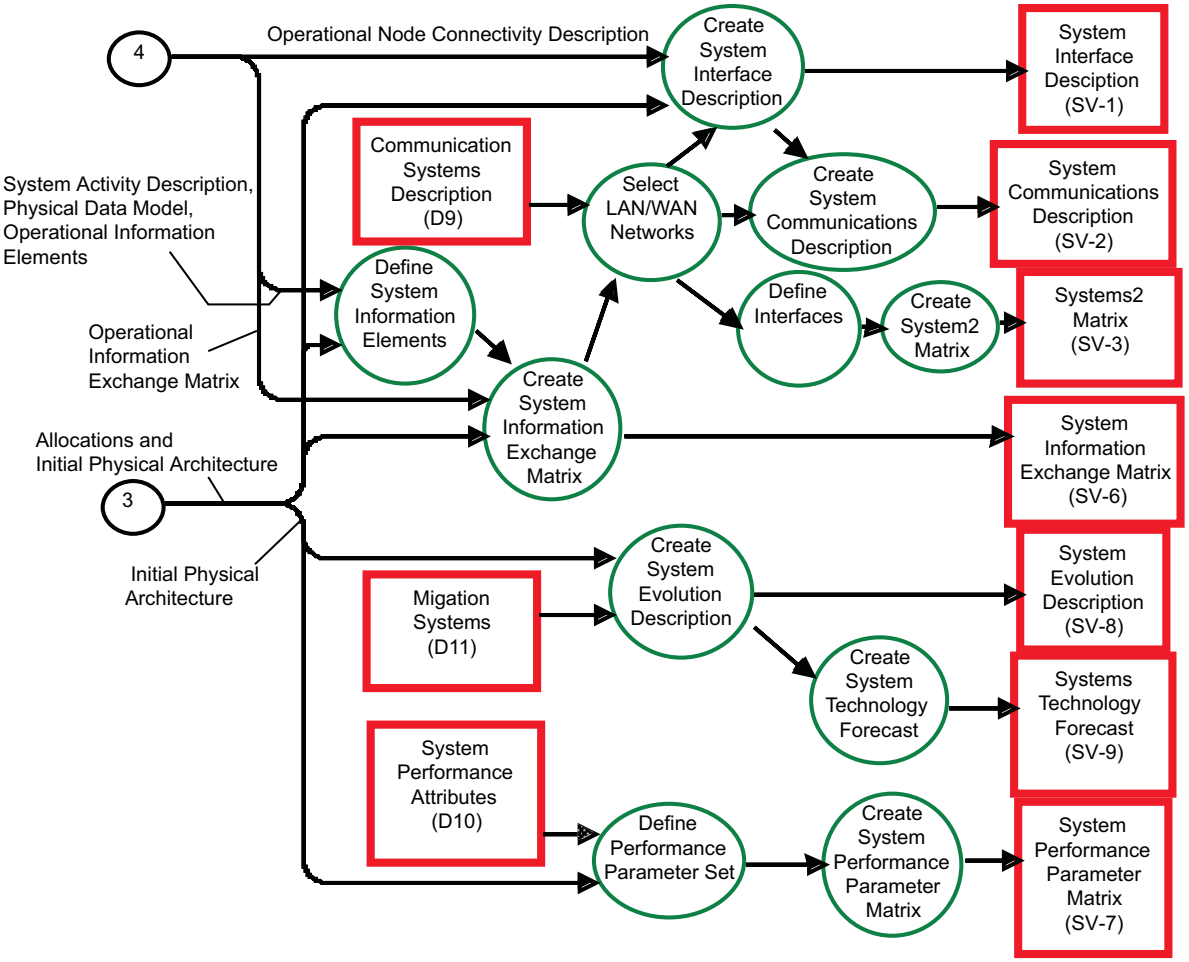


Figure 6. Process model of Stage 5.

we assume that OilCo has been informed about a new technology that allows information to be encoded in a small device and be retrieved by a small radio signal. OilCo believes that such technology can be incorporated in gasoline pumps to make it more convenient for drivers to purchase gasoline via a credit card account. Drivers would need to sign up for the FastPass service and provide to OilCo the standard information contained on the credit card they normally use to purchase gasoline. OilCo would store this information in a Central database and issue the driver a FastPass device in the form of a rear window mounted tag or a key chain. The device would contain a unique code that OilCo could match to the credit card information in the central data base. The following Operational Concept Graphic (Fig. 7) is created by the architect during Stage 1. As part of the analysis of the operational concept, the architect establishes (with the client—OilCo, in this case) the desired behavior of the system.

3.1. Stage 1: Operational Concept

Operational Concept Narrative: The Point of View (POV) taken is that of the Systems Architect responsible for the design of the FastPass implementation. Specifically, the Systems Architect is considering a key thread—the sequence of activities that take place when an individual Driver pulls up at the pump and uses the FastPass system to get gas for his car. (This helps establish the boundaries of the system being considered and for which the architecture will be described. The boundaries are also shown in the operational concept graphic in Fig. 7.)

Assumptions: Real gas stations have multiple bays, with each bay having two or four double-sided pumps. All bays but one are usually in the Self-Service mode. The other one is in Full Service mode and does not include the FastPass service. *For simplicity, we shall consider the operation of a single pump equipped with the FastPass system.*

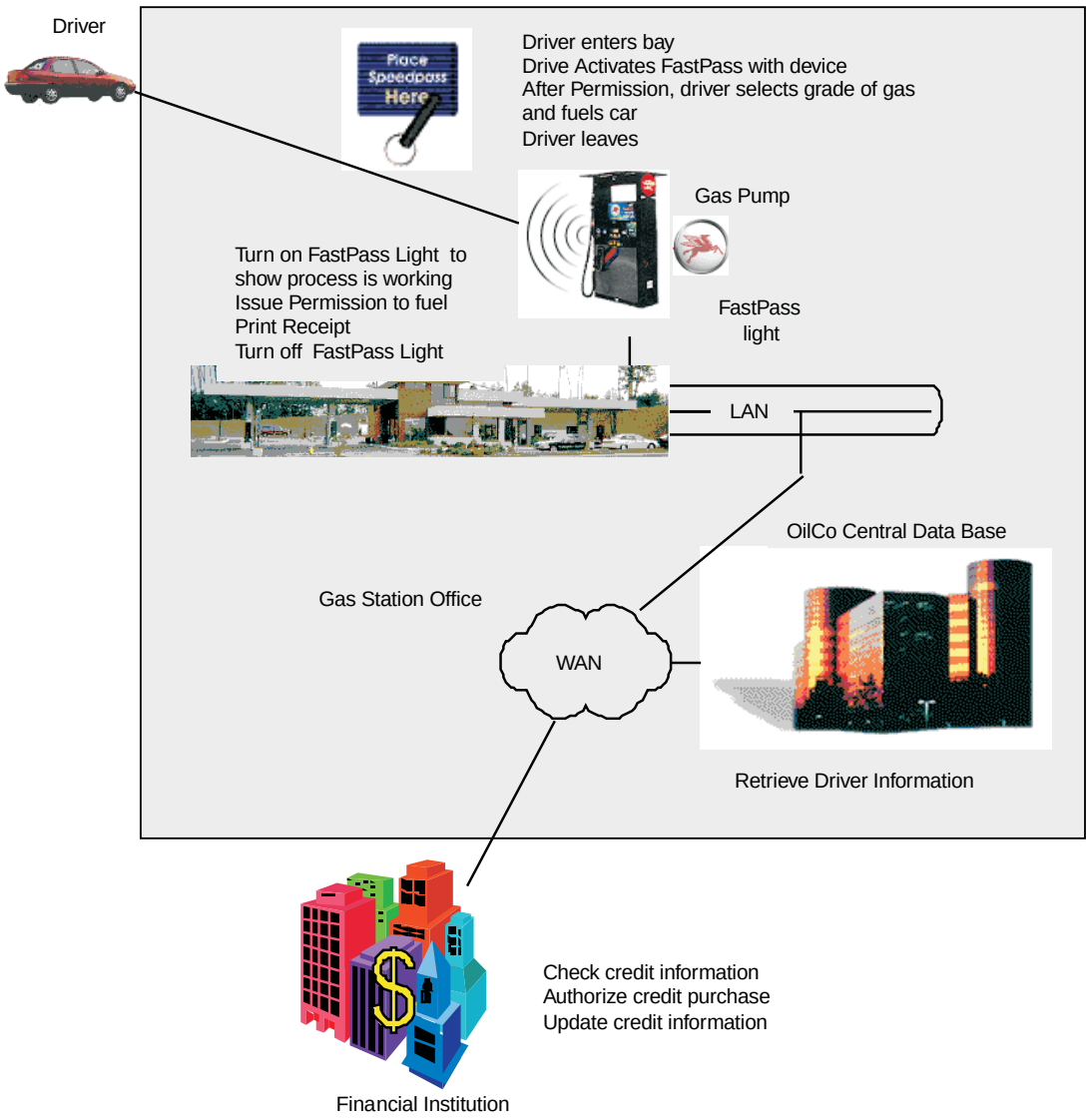


Figure 7. Operational Concept Graphic (OV-1).

The FastPass service is a new alternative to the traditional payment services: pay cash, pay by credit/debit card inside the station, pay by credit/debit card at the pump. As a result, the Systems Engineer need not worry about the details of the electronic accounting system used by the gas station. He need only consider that these functions have been implemented and are available to be parts of the FastPass system.

The process can be described as follows: A Driver pulls up at one of the Self-Serve fuel pumps equipped with FastPass system. If his car is equipped with the FastPass tag, then the sensor on the pump senses its presence and reads the information on the tag. If the Driver has a key-chain tag, he waves the tag in front of the sensor (1–2 in. away), and the sensor reads the information. The sensor lights up.

The information read from the FastPass tag or key chain is decoded at the pump and is then sent from the pump, through the LAN at the gas station, to the FastPass Central database where the relevant credit card information is retrieved. The request for authorization along with the credit card data are then transmitted to the financial institution issuing the credit card. If the request is approved, the fuel pump is enabled and the Driver can pump gas. If the request is denied, the pump is not enabled and the process terminates with the FastPass light going off. (Note that a simpler alternative is for the central database to maintain a current list of invalid credit card numbers, check the request against them, and issue an authorization accordingly.)

If enabled, the Driver *selects the grade of gas* he desires and pumps gas until he turns off the pump by throwing a

switch. Then the cost of the sale is computed at the pump and the amount is transmitted back to the financial institution where it is entered as a charge in the Driver's corresponding credit card account. A receipt is issued at the pump. The data about the sale are entered in the electronic ledger of the gas station. The pump returns to the idle state.

Defining required behavior: From this description, it is possible to infer characteristics of behavior desired in the architecture. We see that the system progresses through several steps or states during the operation of the FastPass system by a customer. In addition, we see that there are several stimuli and responses that take place between the Driver and the system. Specifically, the system has an idle state when it is not in use. When a Driver arrives, he/she presents the FastPass device, and the system acknowledges the detection of the device. The system then goes through a process of retrieving the credit card data from the Central database and obtaining a credit authorization from the "financial institution." It tells the Driver the result of the credit check. There are two possibilities, credit is approved or not approved. If credit is not approved, the Driver may not pump the gas and the pump returns to the idle state. If approval is received, the Driver is instructed to select a grade of gas and to commence pumping. The Driver must respond by selecting the grade and operating the pump including indicating when he/she is finished by "flipping a switch." Finally, the pump computes the total cost of sale, provides a receipt to the Driver, and sends the results to the financial institution. The financial institution acknowledges the receipt and updating of financial accounts to the gas station. The pump returns to the idle state.

Note that it is this description of behavior that is of prime interest to the architect's clients. This is often referred to as a key thread. In addition to producing all of the C4ISR Framework products in a manner that is consistent with this behavior, we will create an executable model that is derived from the architecture that will clearly demonstrate the desired behavior. Indeed, if the executable model does not yield the desired behavior as expressed in the key thread, the architect must modify the architecture design until it does.

3.2. Stage 2: Create the Functional Decomposition and the Organizational Structure

The following functions have been organized in the form of the Universal Joint Task List (UJTL). In an actual DoD case, the architect will have to select the appropriate tasks from UJTL and construct the hierarchy that is appropriate to the domain and that is consistent with the UJTL. In this simple example, the functions that are the basis of the Functional Decomposition are:

1. Validate accounts
   1.1. Sense FastPass
   1.2. Retrieve driver information
   1.3. Validate credit
2. Operate pump
   2.1. Receive authorization
   2.2. Dispense gas
   2.3. Compute cost of sale
3. Prepare billing
   3.1. Request charge
   3.2. Print receipt
   3.3. Update accounts

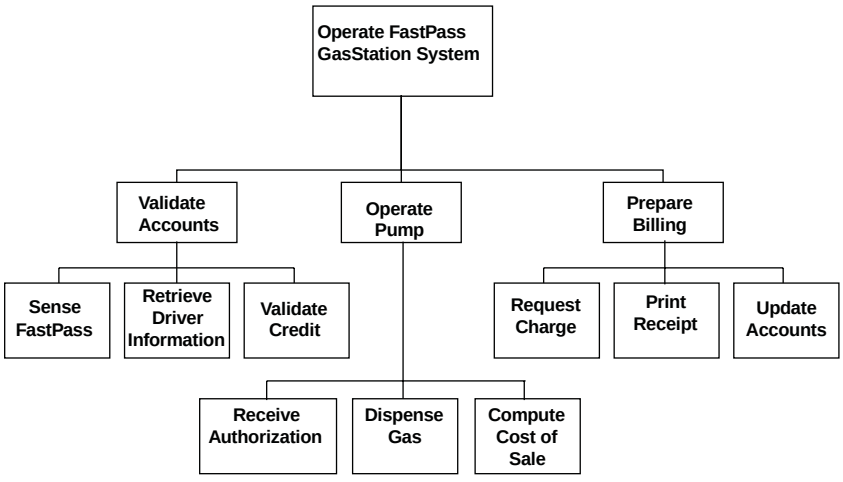


Figure 8. Functional Decomposition.

Table II. Organizations and Assets

Organization	Assets
OilCo	FastPass Central Data Base
Gas Station	Pump, Gas, Ledger
Financial Institution	Financial Account Database
Driver	FastPass Account, Credit Account

The Functional Decomposition is shown in Figure 8. Note that not all of the functions in the UJTL have been used. The architect selects only those functions that are necessary for the purpose of the architecture.

Organizations: For this example, to keep the dimensionality low, only four organizational entities are considered as shown in Table II. Each organizational entity has assets that are the basis for systems in the systems view: The FastPass central database is maintained by OilCo that manages the FastPass system; the Gas Station has a pump, gas, and the electronic ledger for recording sales of gasoline; the Financial Institutions that issue the credit cards used by the Drivers through the FastPass system, and the Driver. Note that, for simplicity, we will consider a single pump. Considering multiple pumps complicates the logic of the operation substantially; we will then have to consider the asynchronous arrival of Drivers, the concurrent operations, etc. While it is doable, it does not provide additional insight.

Operational nodes and operational elements are selected from these organizations (Table III). Operational nodes are graphical objects that will be depicted on the Operational Node Connectivity Description. Each Operational Node will represent one or a collection of operational elements.

The last activity of Stage 2 is to create the Command Relationship Chart (OV-4). In this case we know that OilCo provides a franchise to the Gas Station and provides the FastPass account to the Driver. Both the Driver and the Gas Station have accounts with the Financial Institution. The Driver and the Gas Station Interact with each other. These relationships are depicted in the Command Relationship Chart of Figure 9.

Table III. Operational Nodes and Elements

Operational Node	Operational Elements
Driver	Driver
Gas Station	Pump, Gas Station Office
OilCo	OilCo FastPass Database
Financial Institution	Financial Institution Account Database

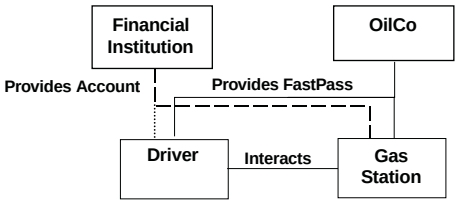


Figure 9. Command Relationship Chart.

3.3. Stage 3: Create Functional Architecture and Initial Physical Architecture

Stage 3 begins with creating the three models that comprise the Functional Architecture: the Activity Model (IDEF0), the Data Model (IDEF1X), and the Rule Model. These also are Framework products Activity Model (OV-5), Logical Data Model (OV-7), and Operational Rules Model (OV-6a), respectively.

One way to begin the Activity Model is to create an External System Diagram (Fig. 10). The perspective of this diagram is standing on the boundary of the system and looking outward at the interactions with activities in the environment of the system. These external systems and interactions are gleaned from the operational concept. The box representing the system is given the index number A0 and the boxes representing the external systems represent the functions these systems perform and are given index numbers A-01, A-02, … As with all IDEF0 boxes, External System activities are expressed as verb phrases.

Figure 10 shows the External System Diagram for the FastPass System. There are two external systems that interact with the FastPass System, the Driver and the Financial Institution. The overall activity of each in relation to the FastPass system is expressed as a verb phrase. The diagram shows that the Driver presents the FastPass Device to the System. This is modeled as a control because it activates the FastPass system. The FastPass system interacts with the financial institution in two ways. First, it sends a financial request in the form of an Authorization Request to the financial institution to determine that the credit of the Driver is okay. The financial institution returns the Authorization request with either an approval or denial. The returned authorization request is modeled as a control to the FastPass system because it determines whether the Driver can purchase gasoline using the FastPass System or not. The second financial request is in the form of a Request for Charging the Driver's credit card account for the purchase and crediting the gas station account with the amount of the sale. The Financial Institution returns a Bank Transaction to the FastPass system with

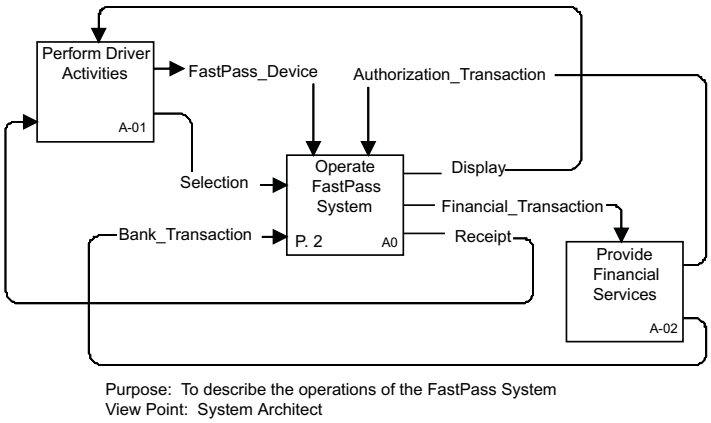


Figure 10. External System Diagram. Purpose: To describe the operations of the FastPass System. Viewpoint: System Architect.

the results of this charging activity. After the Driver has purchased the gasoline, the FastPass system provides a Receipt to the Driver. The System also provides instructions to the Driver via the display. These are modeled as controls to the Driver activity because the display primarily provides direction to the Driver activity.

The External System Diagram is consistent with the operational concept and defines the interfaces between the system and its environment. By itself, it can be used to discuss the operation of the system with the customer to ensure all agree on the system boundary and interactions. Once created, the traditional context diagram of the IDEF0 Activity model is easily obtained by deleting the External Systems from the diagram. The resultant context diagram is shown in Figure 11. The other pages of the IDEF0 model are shown in Figures 12–15.

The Data Model is shown in Figure 16. In general, each Input, Control, and Output represents an entity in the Data Model, although it is permissible to have any one of them represent an attribute of an entity. The selection of the attributes and the relationships that determines the migration of the Keys must be consistent with the activities in the IDEF0 model.

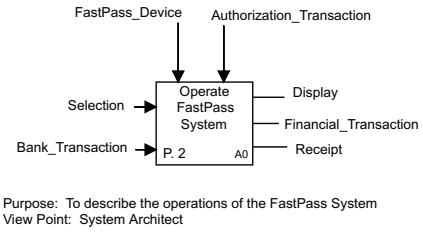


Figure 11. Context diagram for the activity model. Purpose: To describe the operations of the FastPass System. Viewpoint: System Architect.

For brevity, only the Rule Model associated with the Activity "Operate the Pump" is shown. Note that the Rules represent the activation rules for the leaf functions of the IDEF0 model and that the clauses of the rules match the entities and their attributes of the data model as well as the inputs, controls, and outputs of the function for which they apply.

Rule for Activity A22:. Dispense Gas

R22: If Authorization_Transaction.approval = true
And (Selection. "on", Selection.QuantityControl, Selection.Grade are Selected)
Then
(Dispensed_Gas_Data.Grade = Selection.Grade;
Dispensed_Gas_Data.QuantityControl = Section.QuantityControl)
And Display.Content = "When done, turn off the pump";The Grade of Gas is "
+ Dispensed_Gas_data.Grade,
"The dispensed QuantityControl of Gas is "+ Dispensed_Gas_Data.QuantityControl;
Else Display.Content = "Credit Card Purchase not Authorized, Select
Another Payment Type";

The architect creates a State Transition Diagram for the FastPass System as shown in Figure 17. Note that this State Transition Diagram is consistent with both the models and the behavioral description elicited during the domain information gathering stage. The states and the transitions between them can be obtained by tracing a key thread through the IDEF0 model starting with the arrival of a Driver at the pump.

The architect next turns to the process of allocating the activities of the functional architecture to the operational elements and nodes that were selected in Stage 2

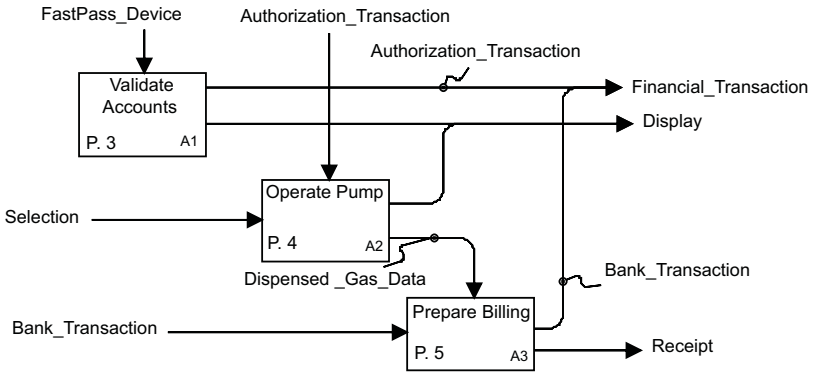


Figure 12. First level of decomposition.

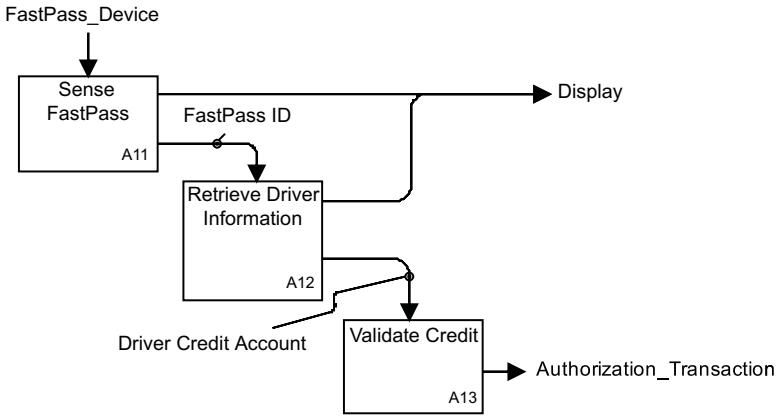


Figure 13. Validate accounts page.

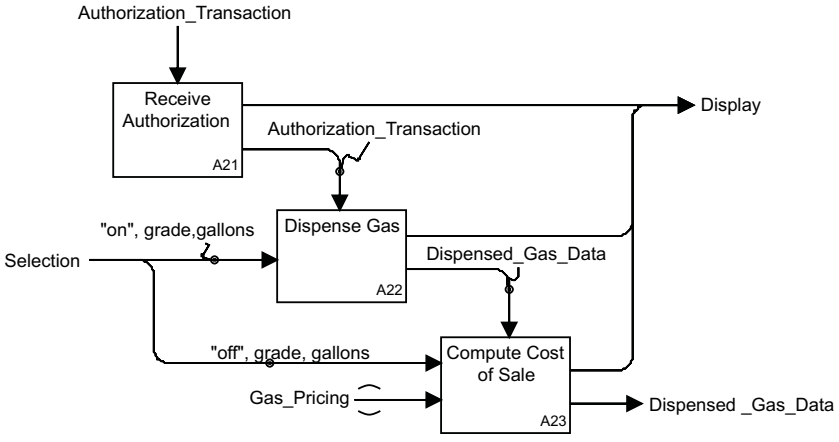


Figure 14. Operate pump page.

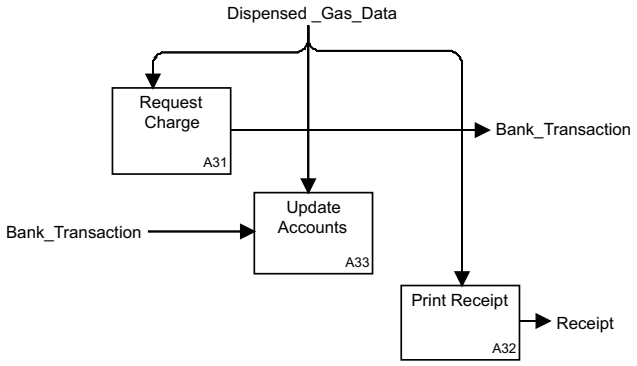


Figure 15. Prepare billing page.

and selecting each operational element to perform the activities. Table IV shows the allocation of activities to the operational elements. Notice that the nine leaf activities of the Activity model have been assigned to four operational elements. "Update Account" and "Validate Credit" are each assigned to two operational elements. Recall that the financial institution is outside the boundary of the system, but interacts with it. Thus, the pump is responsible for the "Validate Credit" activity, although it is accomplished by the financial institution. Similarly, the "Request Charge" will cause the financial institution to "Update Accounts" and will return the

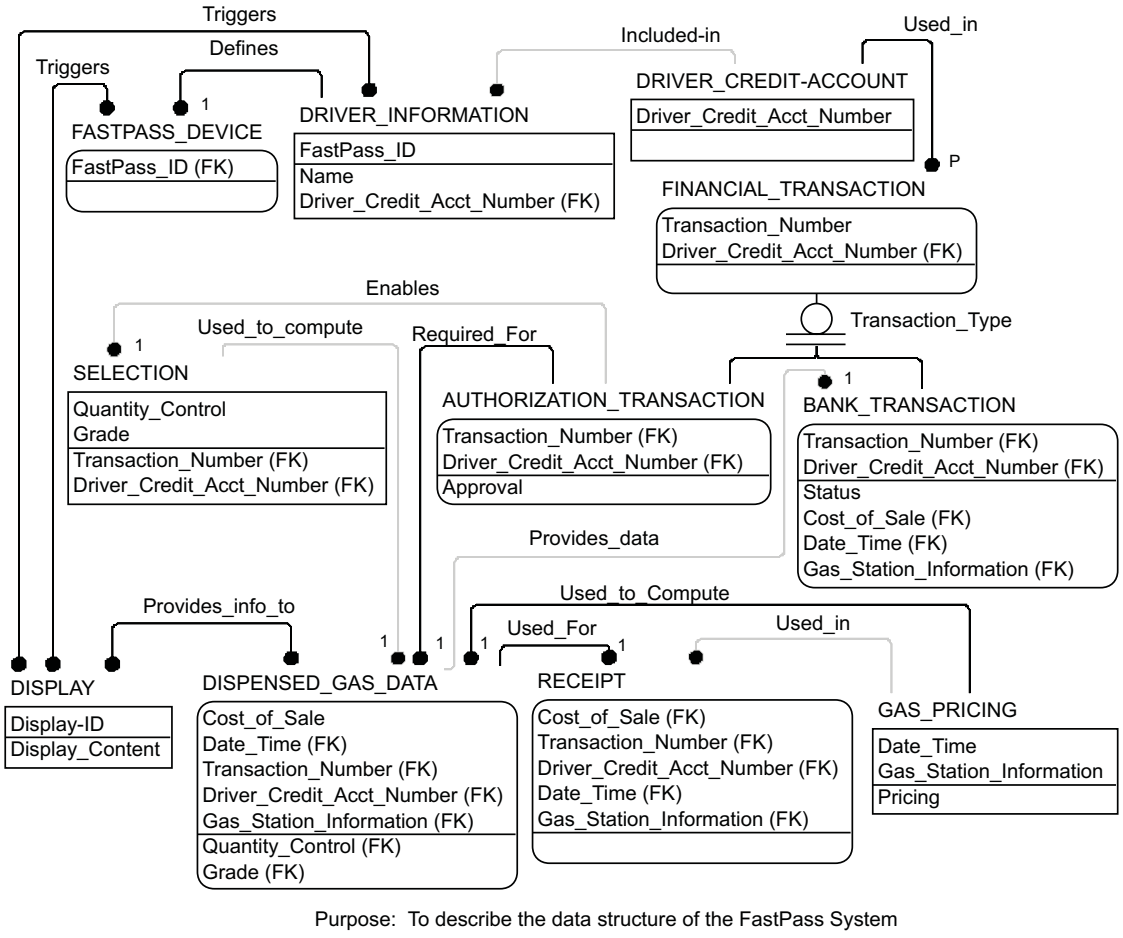


Figure 16. Logical Data Model (OV-7). Purpose: To describe the data structure of the FastPass System.

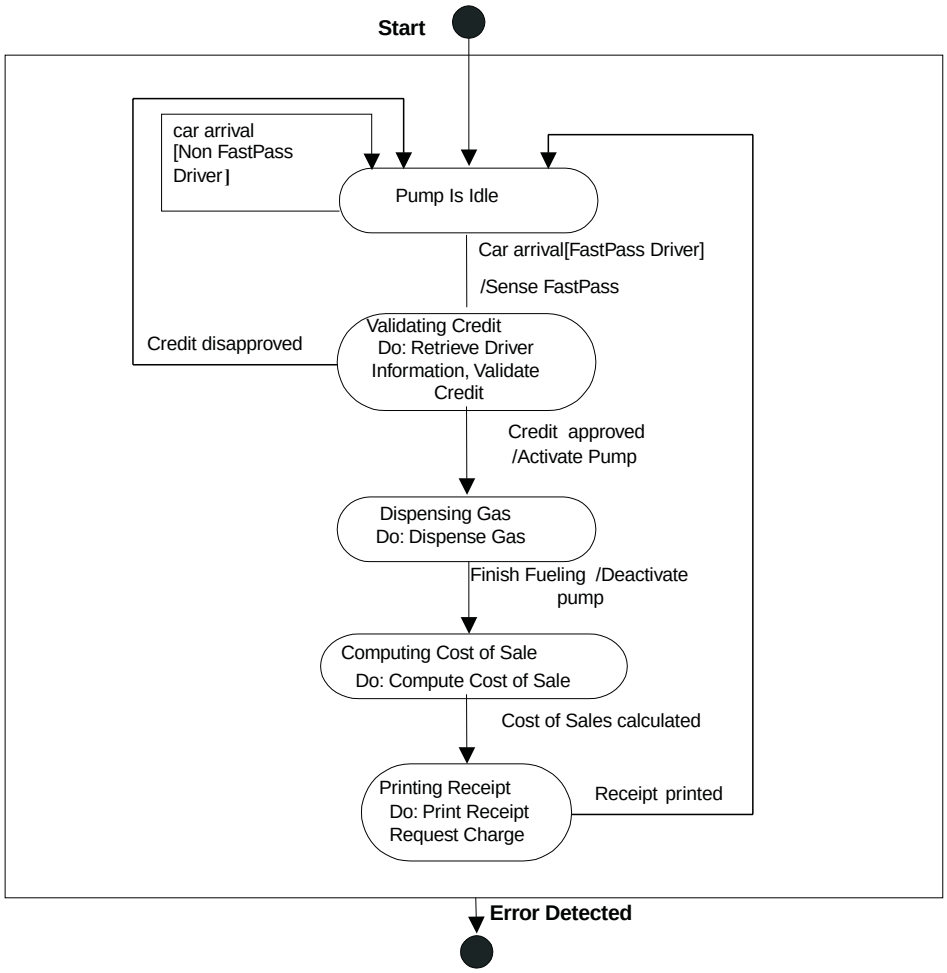


Figure 17. State Transition Diagram (OV-6b).

result to the Gas Station Office where the ledger will be updated.

The architect also will allocate the Operational Activities to the systems which are the assets that the operational elements possess. This is done by matching the Operational Activities to the System Functions the systems can perform. Table V shows the type of information available to the architect (gathered in Stage 0) to support this allocation process. The result of this allocation process is shown in Table VI.

Table IV. Allocation of Operational Activities to Operational Elements

Operational Element	Activities
Financial Institution	(Update Account) (Validate Credit)
Gas Station Office	Update Accounts
OilCo	Retrieve Driver Information
Pump	Sense FastPass Request Charge Receive Authorization Dispense Gas Print Receipt Compute Cost of Sale Validate Credit

The last processes of Stage 3 involve creating two forms of the initial physical architecture. In the first form, the architect uses the operational nodes to construct the skeleton of the Operational Node Connectivity Description. Needlines are connected between the operational nodes indicating that operational information elements are passed between the operational nodes. Figure 18 shows the skeleton of the Operational Node Connectivity Description.

Finally, the architect creates the initial physical architecture composed of one or more diagrams showing system nodes with systems, system ele-

Table V. System Elements, System, and System Functions

System	Element	Functions
Driver	FastPass Tag	Provide FastPass Tag Select Option
Pump	FastPass Sensor	Sense FastPass Tag and Decode FastPass ID
	Pump Control Unit	Control Operation Sense Selection Request Authorization Request Charge
	Message Display (at Pump)	Display Message
	Receipt Printer	Print Receipt
	Calculator	Compute Cost of Sale
	Gas Nozzle Valve	Dispense Gas
	Communication Unit	Receive/ Transmit Signal
Gas Station Office Database	Sales Database	Record Transaction
	Communication Unit	Receive/ Transmit Signal
FastPass Central Database (OilCo)	FastPass Central Database	Retrieve Driver Information
	Communication Unit	Receive/ Transmit Signal
Financial Institution' Database	Account Database	Issue Authorization Manage Database
	Communication Unit	Receive/ Transmit Signal

Table VI. Allocation of Operational Activities to System Functions

Activities	System Functions
Sense FastPass	Provide FastPass Tag Sense FastPass Tag and Decode FastPass ID
Retrieve Driver Information	Sense FastPass Tag and Decode FastPass ID Retrieve Driver Information
Validate Credit	Request Authorization Display Message Issue Authorization
Receive Authorization	Control Operation
Dispense Gas	Sense Selection Display Message Dispense Gas
Compute Cost of Sale	Display Message Compute Cost of Sale
Request Charge	Request Charge
Print Receipt	Print Receipt
Update Account	Record Transaction Manage Database

ments and system components, and the communications links that connect them. The architect uses the Operational Node Connectivity Description as a guide. These Operation Nodes with Assets are represented as system nodes in the initial physical architecture. Figure 19 shows the initial physical architecture for the FastPass system.

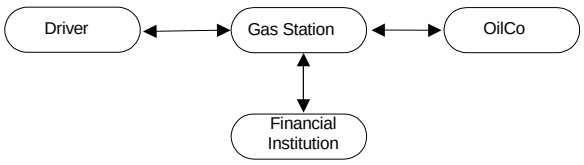


Figure 18. Operational nodes and needlines.

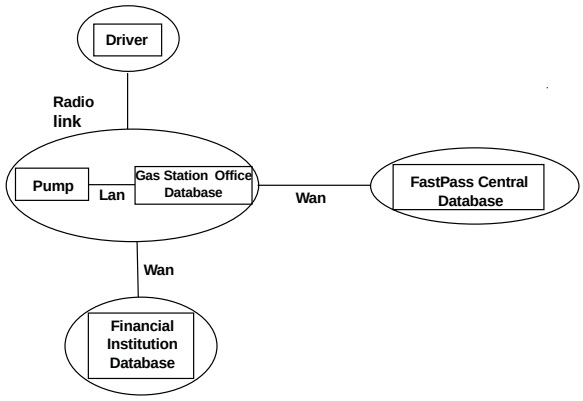


Figure 19. Initial physical architecture.

Table VII. Operational Elements

Operational Information Element	Producing Operational Element
Authorization_Transaction.Approval	Financial Institution
Bank_Transaction.complete	Financial Institution
Dispensed Gas Data	Pump
Display	Pump
Authorization_Transaction.request	Pump
Driver Information	OilCo
Grade of Gas (Selection)	Driver
FastPass Device	Driver
Quantity Control (Selection)	Driver
Receipt	Pump
Bank_Transaction.request	Pump
FastPass ID	Pump

3.4. Stage 4: Complete Operational Architecture Views and Create System Activity Models

During Stage 4, the architect completes the Operational Architecture View Products and produces the physical manifestation of the Functional Architecture into the Systems Functionality Description and a Physical Data Model. The first step in this stage is to define the Operational Information Elements that will be represented in the Operational Node Connectivity Description (OV-2) and the Operational Element Exchange Matrix (OV-3). The main source for the operational elements is the Logical Data Model (OV-7). Each entity is a candidate for an Operational Information Element. These are logical information entities that flow over needlines between operational elements and nodes. The operational nodes and elements have been defined in Stage 3 and the operational activities have been assigned to them. It is straightforward to determine the entities that flow between the operational element. Table VII lists the Operational Information Elements and the Operational Elements that produce them. Note that in some cases the Operational Element consists of an attribute of an entity in the Logical Data Model. For example, Grade of Gas is an attribute of the entity Selection. The Banking Transaction has been added. It represents the information that is sent from the financial institution to the Gas Station Office as part of the activity "Update Accounts."

It is now possible to complete the Operational Node Connectivity Description (OV-2) as shown in Figure 20. This figure shows the operational nodes and the needlines. Note that each operational node contains a window that shows the operational activities that are performed at the node. In addition, there is a window for each needline that shows the operational information elements that are flowing between the operational nodes. One can view this product as a morphed version of the IDEF0 model in which all of the leaf activities have been clustered into their assigned operational nodes and the flows between the activities have been bundled into needlines.

The information contained in the functional architecture models is also reflected in the Operational Information Exchange Matrix (OV-3). Each row of the matrix specifies several characteristics of one of the operational information elements. These characteristics include the name and several parameters about its content, plus list the Operational Element and the operational activity that produces it and the operational

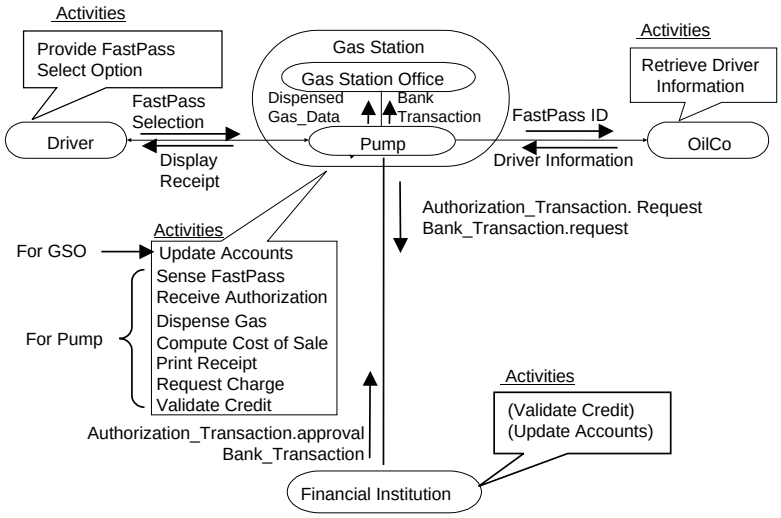


Figure 20. Operational Node Connectivity Description (OV-2).

Table VIII. Operational Information Exchange Matrix (OV-3)

Information Description				Information Source		Information Destination	
Operational Information Element	Media	Size	Unit	Operational element	Activity	Operational element	Activity
FastPass Device	Micro wave	8	Number	Driver	N/A	Pump	Sense FastPass
FastPass ID	Data	8	Number	Pump	Sense FastPass	OilCo	Retrive Driver Information
Grade of Gas	Data	1	Number	Driver	N/A	Pump	Dispense Gas
Quantity Control	Data	10	Number	Driver	N/A	Pump	Compute Cost of Sale
Authorization Transaction. Approval	Data	8	Number	Financial Institution	Validate Credit	Pump	Receive Authorization
Banking Transaction	Data	10	Number	Financial Institution	(Update Accounts)	Gas Station Office	Update Accounts
Authoization Requestt	Data	8	Number	Pump	Validate Credit	Financial Institution	(Validate Credit)
Request for Charge	Data	10	Number	Pump	Request Charge	Financial Institution	Update Accounts
Driver Information	Data	9	Number	OilCo	Retrieve Driver Information	Pump	Validate Credit Request Charge Print Receipt
Display	Data	19	Number	Pump	Sense FastPass Operate Pump	Driver	N/A
Dispensed Gas Data	Data	19	Number	Pump	Compute Cost of Sale	Gas Station Office	Update Accounts
Receipt	Docu ment	2x4 "	Paper string	Pump	Print Receipt	Driver	N/A

element and activity that receives it. The Operational Information Exchange Matrix is shown in Table VIII.

Having completed all of the Operational Architecture view Products, the architect turns to completing the System Architecture view. At this point it is easy to produce the System Function Traceability Matrix (SV-3) using the allocation information created in Stage 3. This matrix is presented in Table IX. Note the many-to-many relationship that can exist between Operational Activities and System Functions. For example, the Operational Activity "Retrieve Driver Information" will be accomplished by two system functions and the System Function "Display Message" will support four Operational Activities. Also note that an indexing scheme has been applied to the System Functions so that each one is associated with the system that performs the function. This grouping will aid in the development of the System Functionality Description.

The next step is to complete the System Functionality Description (SV-4), which is an Activity Model based on Data Flow Diagrams. This model will have the System Functions as its transformations. The architect uses the IDEF0 model and the System Function Traceability Matrix to create the Data Flow Diagram.

The Context Diagram is shown in Figure 21. Notice that it preserves the original system boundary with the Driver and the financial institution outside the system but interacting with it as Terminators.

The first level of decomposition is shown in Figure 22. The decomposition principle was based on grouping system functions associated with the major systems of the architecture. Thus there is a transformation for the Pump system, a single transformation for the OilCo Central system, and a single transformation for the Gas Station Office System. Data Stores are shown connected to the latter two transformations that represent the OilCo Central Data Base and the Gas Station Office Ledger, respectively.

The decomposition of the pump system functions is shown in Figure 23. It is composed of the eight system functions listed in the System Function Traceability Matrix for the Pump. The architect must maintain a consistent mapping between the IDFE0 activity model of the Operational Architecture view and the System Functionality Description since they are both activity models of the same architecture, one from an operational point of view and the other from a systems point of view. This means that, for each System Function that has a one-to-one mapping to an Operational Activity, inputs and controls of the IDEF0 activity should map to the inputs of the transformation in the system model and outputs of the IDEF0 activity should map to the outputs of the transformation. If there is a one-to-many mapping from the operational activity to the system functions, then the inputs and outputs to the aggregate of the system functions should map to the input, controls, and outputs of the operational activity. The system functions (transformations) Sense Selection and Display are examples of this one-to-many mapping. On the other hand, if the mapping from the operation activities to the

Table IX. System Function Traceability Matrix (SV-3)

		Operational Activities								
System	System Functions	Sense FastPass A11	Retrieve Driver Information A12	Validate Credit A13	Receive Authorization A21	Dispense Gas A22	Compute Cost of Sale A23	Request Charge A32	Print Receipt A33	Update Account A34
Driver	Provide FastPass Tag	o								
	Select Option					o				
Pump	Sense FastPass Tag 1.1	o	o							
	Request Authorization 1.2			o						
	Display Message 1.3	o		o	o	o	o			
	Sense Selection 1.4					o	o			
	Dispense Gas 1.5					o				
	Compute Cost of Sale 1.6						o			
	Request Charge 1.7							o		
	Print Receipt 1.8								o	
Gas Station Office Database	Record Transaction 2									o
FastPass Central Database	Retrieve Driver Information 3		o							
Financial Institution Database	Manage Database									o
	Issue Authorization			o						

system function is many-to-one, as is in the case of the system function Display, then the aggregate of the inputs, controls, and outputs in the operational activity should map to the inputs and outputs of the single system activity. The choice of a single termination for Display reflects the architect's decision to have a single display system element, such as an LCD, that tells the customer the FastPass Device has been decoded, the FastPass account data has been retrieved (or is not available), the credit has been (or not) validated, and the cost of the sale.

The architect also creates the Physical Data Model (SV-11). It describes the physical manifestation of the entities in the Logical Data Model as it describes the actual messages that are flowing in the Data Flow Diagram and the data that are in the Data Stores. A tabular format is used to list each message or record and the fields of each. Table X shows the Physical Data Model for the FastPass system.

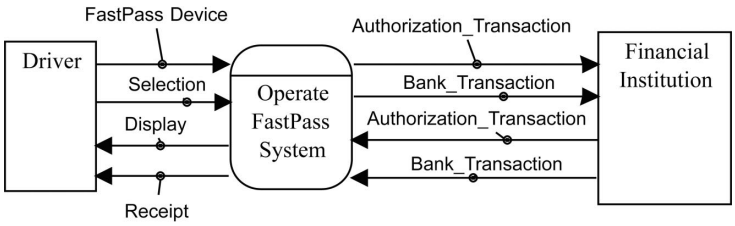


Figure 21. Context Diagram. Purpose: To describe the Systems Functions of the FasPass System. Viewpoint: System Architect.

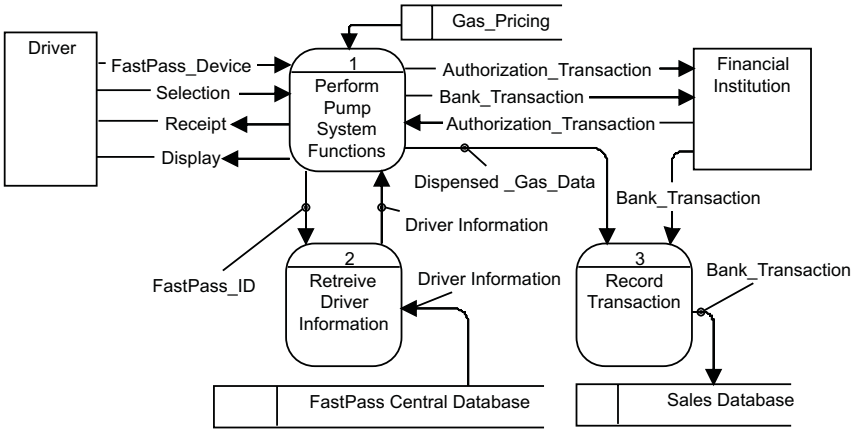


Figure 22. First level of decomposition.

3.5. Stage 5: Complete System Architecture Views

In Stage 5, the architect completes the System Architecture view products, drawing on the knowledge and information created in the earlier stages and adding new refinements such as details of the communications architecture, descriptions of interfaces, and future migrations, capabilities, and technologies. Again, the architect must ensure that consistency is maintained across all products.

The architect begins by defining the System Information Elements. These are consistent with both the

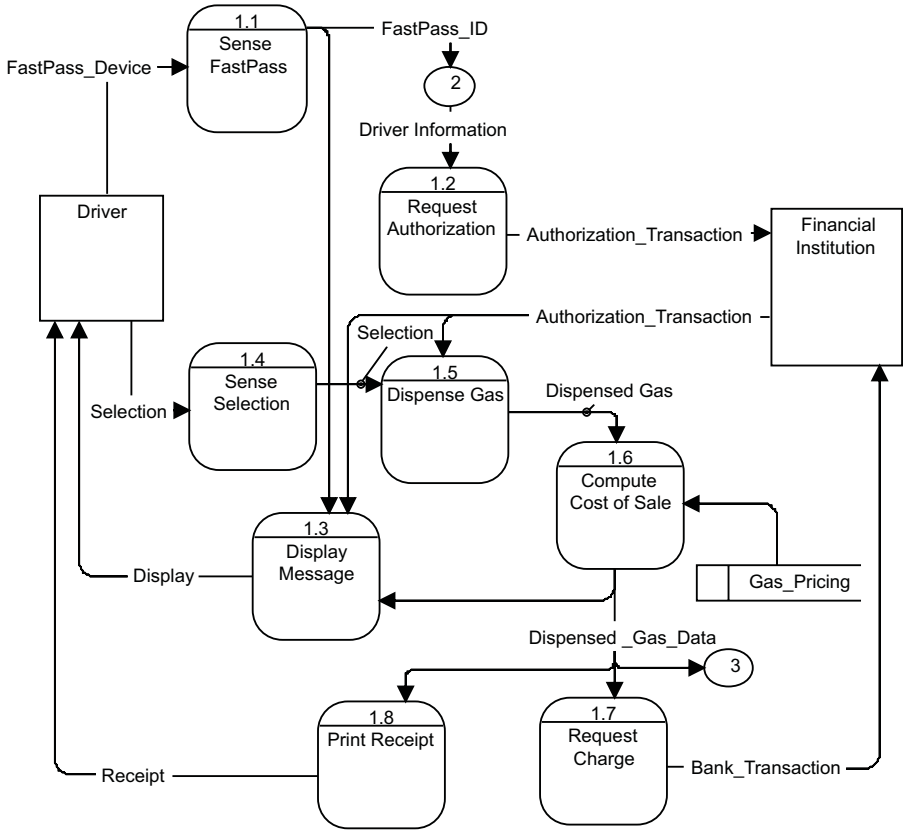


Figure 23. Decomposition of the "Perform Pump System Functions" transformation.

Table X. Physical Data Model

Messages:

Records	Fields	Format
Selection	- Quantity Control - Grade of Gas	Numeric(10) Numeric(1)
FastPass Device Data	- FastPass ID (Encrypted)	Numeric(8)
Dispensed Gas	- Driver Credit Account Number - Transaction_Number - Quantity Control - Grade of Gas	Numeric(8) Numeric(10) Numeric(10) Numeric(1)
Bank_Transaction	- Gas Station Office Account - Status - Cost of Sale - Transaction_Number	Numeric(16) Boolean Numeric(10) Numeric(10)
Authorization_Transaction	- Transaction_Number - Driver Credit Account - Approval Content	Numeric(10) Numeric(10) Boolean
Dispensed Gas_Data	- Driver Credit Account Number - Transaction_Number - Cost of Sale - Quantity Control - Grade of Gas	Numeric(8) Numeric(10) Numeric(10) Numeric(10) Numeric(1)
Bank_Transaction.Request	- Driver Credit Account - Gas Station Office Account - Cost of Sale - Transaction_Number	Numeric(16) Numeric(16) Numeric(10) Numeric(10)
Receipt	- Gas Station Name - Gas Station Address - Date - Driver Credit Account - Name - Grade of Gas - Quantity Control - Cost of Sale	Char(40) Char(40) Numeric(6) Numeric(16) Char(40) Numeric(1) Numeric(10) Numeric(10)

Data Stores:

Records	Fields	Format
Driver Information (FastPass Central Data Base)	- FastPass ID - Name - Driver Credit Account Number	Numeric(8) Char(40) Numeric(16)
Sales Data Base	- Gas Station Office Account - Sales History	Numeric(16) Numeric(10)
Gas Station Office (Gas_Pricing)	- Gas Station Office ID - Gas Station Name - Gas Station Address - Gas Unit Price	Numeric(16) Char(40) Char(40) Numeric(10)

Display:

Records	Fields	Format
Display	- Display ID - Message	Numeric(2) Char(40)

Table XI. System Information Elements

System Name	Content	Media	Data/Media Format	Security	Frequency
Driver	Selection	Data	ASCII	Plain	Dynamic
	FastPass Tag	Microwave	Radio Signal	Secure	Dynamic
Pump	FastPass ID	Data	ASCII	Secure	Dynamic
	Dispensed Gas_Data	Data	ASCII	Plain	Dynamic
	Cost of Sale	Data	ASCII	Plain	Dynamic
	Request for Charge	Data	ASCII	Secure	Dynamic
	Driver Credit Account	Data	ASCII	Secure	Dynamic
	Receipt	Document	Text	Plain	Dynamic
	Message	Display	Text	Plain	Dynamic
Gas Station Office Database	Bank_Transaction	Data	ASCII	Plain	Dynamic
FastPass Central Database	Driver Information	Data	ASCII	Secure	Dynamic
Financial Institution Database	Authorization_ Transaction	Data	ASCII	Secure	Dynamic
	Bank_Transaction	Data	ASCII	Secure	Dynamic

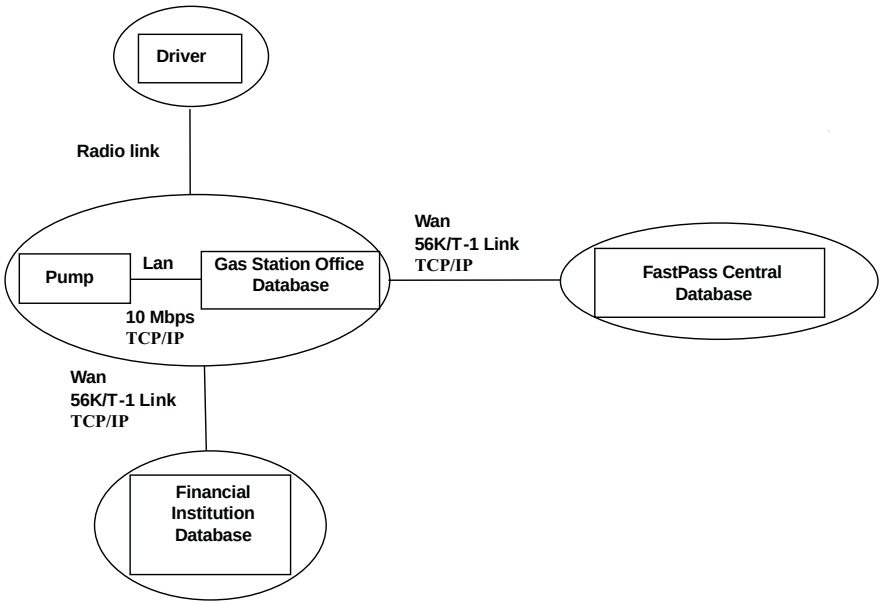


Figure 24. LAN/WAN selections.

Logical Data Model and the Physical Data Model. The architect specifies attributes for each System Information Element as shown in Table XI.

The architect uses the information about the System Information Elements to select the types of LANs and WANs for the architecture. These are added to the Initial Physical Architecture as shown in Figure 24. The architect has sufficient information to create the System Interface Description (SV–1). An Intra-System view (Node-Edge to Node-Edge) is shown in Figure 25. If necessary, the architect can expand the communication system architecture of the System Interface Description and produce the Systems Communications Desciption (SV-2). An example is shown in Figure 26.

The Communications System helps specify the interfaces between the System Nodes, Systems, System Elements, and System Components. The System^2 Matrix (SV-3) is a compact product that tabulates this aspect of the architecture as shown in Figure 27. The architect determines the types of interfaces and creates a key that is used to fill out the matrix.

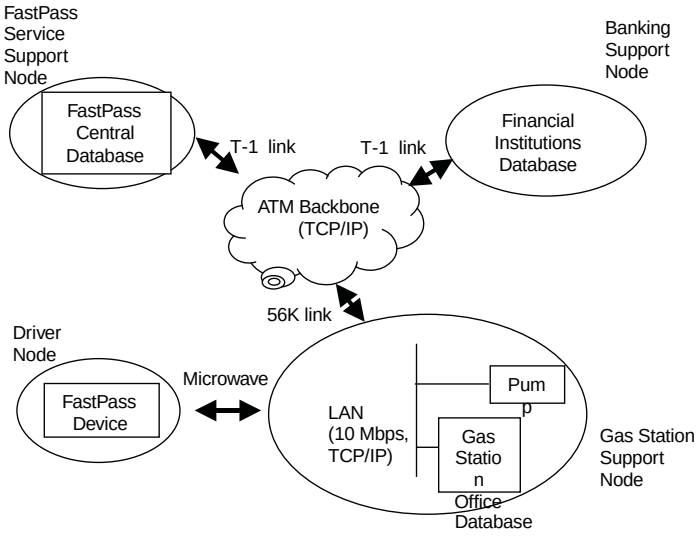


Figure 25. System Interface Description (SV-1).

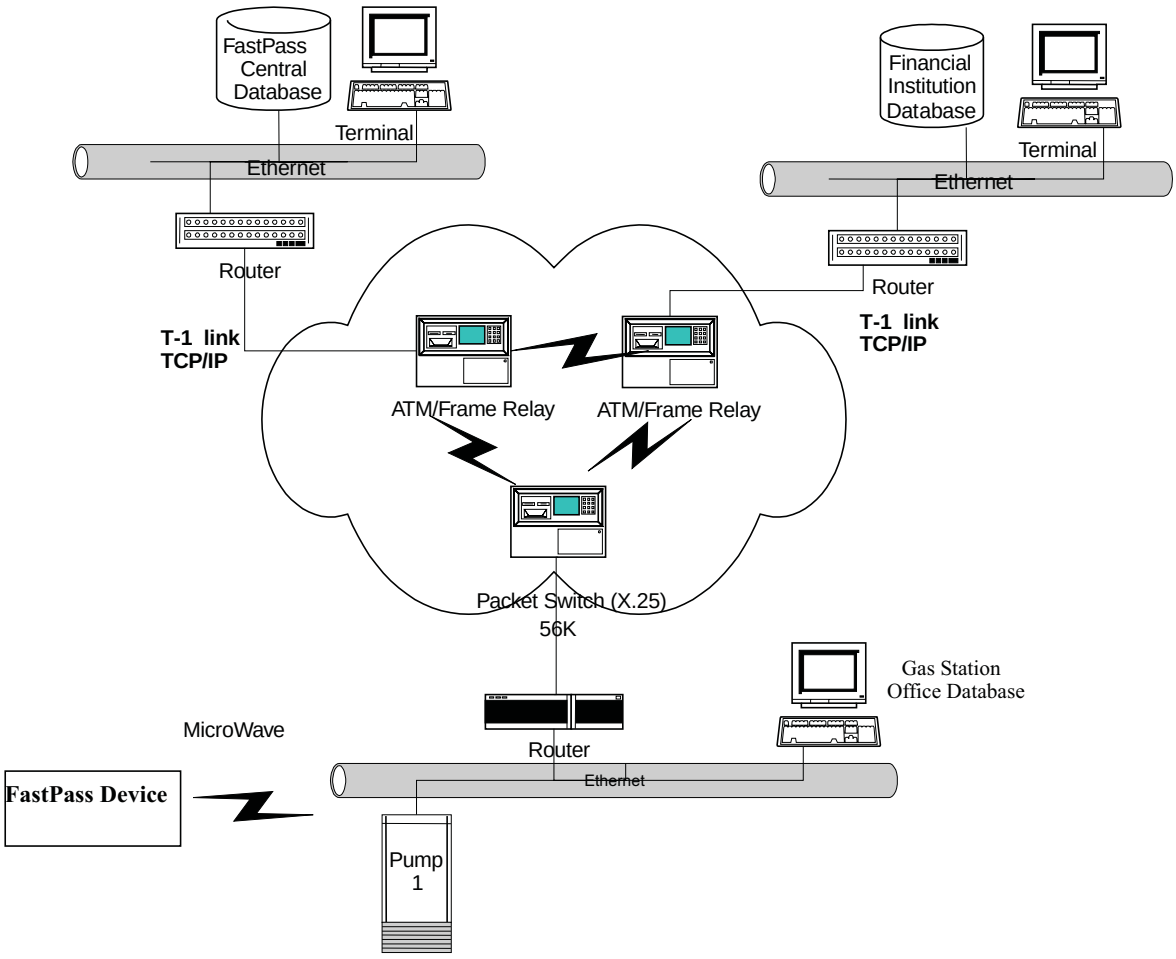


Figure 26. System Communications Description (SV-2).

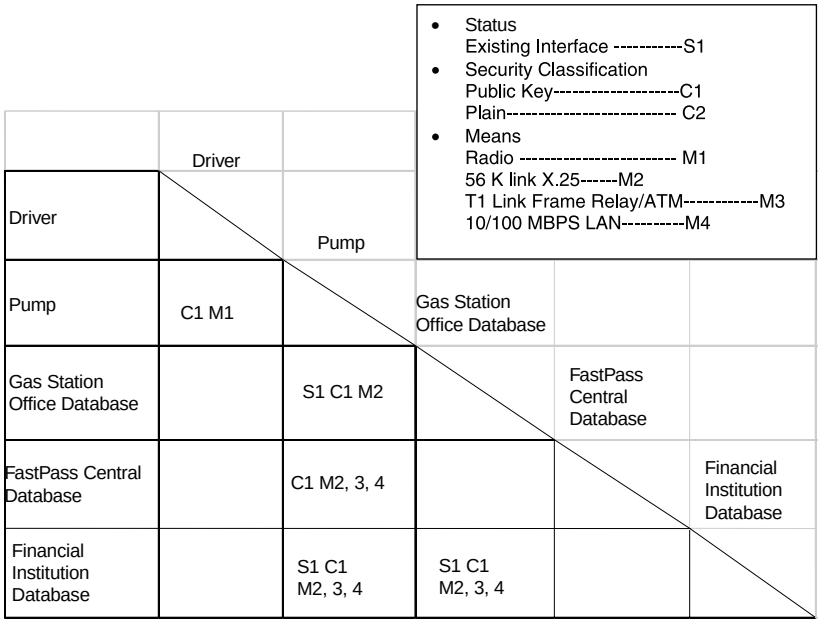


Figure 27. System^2 Matrix (SV-3).

The System Information Exchange Matrix (SV-6) provides in tabular form much of the information provided in the System Functionality Description and the Physical Data Model. Like the Operational Information Exchange Matrix, it associates the System Information Elements with the System Functions of the system activity model, the Data Flow Diagram. It also provides important attributes of each System Information Element. The format of the columns of the matrix is Input, System Function, Output. An example of the System Information Exchange Matrix is shown in Table XII. This product must be consistent with all other products. For example, the requirements for security must match the interface description of the System^2 Matrix.

If required, the architect may also create the System Performance Parameter Matrix (SV–7), the System

Table XII. System Information Exchange Matrix

Input						System Function	Output					
System Name	Content	Media	Data/ Media For- mat	Sec urity	Freq- uency		System Name	Content	Media	Data/Me dia Format	Sec- urity	Freq - uen cy
Driver	FastPass Tag	Microw ave	ASCII	Secure	Dyna mic	Sense Fast-Pass Tag	Pump	Fast-Pass ID	Data	ASCII	Sec- ure	Dyn amic
Pump	Activate/ Deactivate Signal	Data	Binary	Plain	Dyna mic	Display Message	Pump	Message	Dis- play	Text	plain	Dyn amic
	Dispensed Gas Data	Data	ASCII	Plain	Dyna mic							
Pump	Cost of Sale	Data	ASCII	Plain	Dyna mic	Print Receipt	Driver	Receipt	Docu- ment	Text	plain	Dyn amic
FastPass Central Database	Driver Information	Data	ASCII	Secure	Dyna mic							
Driver	Selection	Data	ASCII	plain	Dyna mic	Dispense Gas	Pump	Activate/ Deactivate Signal	Data	Binary	Plain	Dyn amic
Financial Institution Database	Authoriza- tion. Approval	Data	ASCII	Secure	Dyna mic							
Pump	Activate/ Deactivate Signal	Data	Binary	Plain	Dyna mic		Pump	Dispensed Gas Data	Data	ASCII	plain	Dyn amic
	Selection	Data	ASCII	plain	Dyna mic		Gas Station Office Data base	Dispensed Gas Data	Data	ASCII	plain	Dyn amic
Pump	Dispensed Gas Data	Data	ASCII	plain	Dyna mic	Compute Cost of Sale	Pump	Cost of Sale	Data	ASCII	Plain	Dyn amic
Pump	Dispensed Gas Data	Data	ASCII	plain	Dyna mic	Record Trans- action	Gas Station Office Data base	Trans- action	Data	ASCII	plain	Dyn amic
Financial Institution	Bank_ Transaction	Data	ASCII	Secure	Dyna mic							
Pump	Driver Information	Data	ASCII	Secure	Dyna mic	Request Author- ization	Fin ancial Institut ion Data base	Authorizati on. approval	Data	ASCII	Secure	Dyn amic
Pump	Cost of Sale	Data	ASCII	Plain	Dyna mic	Request Charge	Fin ancial Institut ion Data- base	Request for Charge	Data	ASCII	Secure	Dyn amic
	Driver Information	Data	ASCII	Secure	Dyna mic							
FastPass Central Database	FastPass ID	Data	ASCII	Secure	Dyna mic	Retrieve Driver Inform- ation	Pump	Driver Inform- ation	Data	ASCII	Secure	Dyn amic

Table XIII. System Performance Parameter Matrix

System Name	Performance Parameters	Base Line	Objective
Pump	FastPass detection accuracy	N/A	99.9%
FastPass Central Database	Driver Information verification time	N/A	Limit less than 3 seconds
Financial Institution Database	Credit Validation Time	3 seconds	N/A

Table XIV. System Technology Forecast

Technology Area & Capabilities	0-6 Months	6-12 Months	12-18 Months	18+ Months
Security	Public key algorithm			
Distributed Heterogeneous Database	Middleware and/or proprietary interface		Dynamic active DBMS	
Satellite Communication	World wide DAMA (Demand Access Multiple Access) ground terminal migrate into WAN			
OilCo Subscriber Roaming	System wide roaming across the boundary of home service area			International roaming across the boundary of domestic service area Heterogeneous DBMS

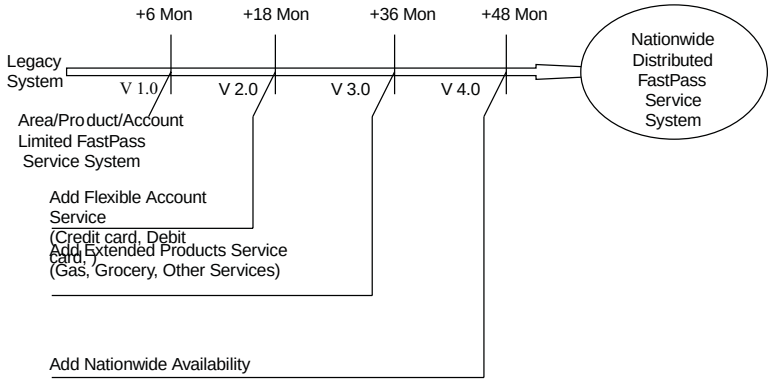


Figure 28. System Evolution Description (SV-8).

Evolution Description (SV-8), and the System Technology Forecast (SV–9) as shown in Tables XIII and XIV and Figure 28.

4. SYNTHESIS OF THE EXECUTABLE MODEL

It is possible to create an executable model of the architecture using the information and models produced during Stage 3 of the process. In the first article of this series [Levis and Wagenhals, 2000], we described the creation of the executable model as the synthesis phase, in which the information developed in the analysis phase is synthesized into an executable model. In this section, we describe how to synthesize such a model and illustrate how it can be used to verify that the behavior of the architecture matches the desired one. The discrete event system modeling paradigm is used, and we use Colored Petri Nets as the modeling method for creating the discrete event system ex-

ecutable model of the architecture. The choice of modeling paradigm is dictated by the nature of the processes being modeled—in this case, decision processes and asynchronous events.

4.1. CPN Colored Petri Nets

Information Systems are dynamic in nature. Events occur that trigger the execution of functions, and many functions can be executed concurrently. An executable representation of the system illustrates the dynamic behavior and permits the evaluation of time-related measures of performance. It also enables the formal analysis of the model to determine its logical and behavioral characteristics. There exist some several graphical modeling approaches that allow a dynamic representation of discrete event systems. Colored Petri Nets (CP nets) [Jensen, 1992], Finite State Machines, Behavior Diagrams, and Queuing Nets are examples of such approaches. They can be used directly to model a discrete event dynamical system representation of an information architecture.

The basic conversion of an Activity model into a Petri Net was described in Levis and Wagenhals [2000]. Petri Nets consist of places, transitions, directed arcs, and tokens. In Colored Petri Nets, the tokens are distinguishable; they are characterized by their color: An attribute vector is associated with each token. The assignment of values to the attributes from their respective domains specifies the color of the token. Color sets are associated with places; they specify which token can reside in that place. Complex enablement conditions can be specified on the arcs between input places and transitions. Each input arc inscription specifies the number and type of tokens that need to be in the place for the transition to be enabled. The output arc inscriptions indicate what tokens will be generated in an output place when the a transition fires. Furthermore, guard functions associated with transitions are allowed. These guard functions specify additional conditions that must be satisfied, i.e., in addition to those inscribed on the arcs, for a transition to be enabled. Code segments can be associated with transitions. These code segments can represent the function modeled by the transition and complement the output arc inscriptions.

Each Colored Petri Net model has a Global Declaration node associated with it that contains the definitions of all Color Sets and their associated domains and the definition of variables. It becomes apparent then that much of the data in the data dictionary of an architecture appears in the global declaration node of the Colored Petri Net model.

The use of Colored Petri Nets to develop an executable model from the Structured Analysis models can be described as follows.

4.2. Implementation Sources

The executable model is derived from four static models of the architecture: the activity model (IDEF0), the data model (IDEF1X), the rule model, and the state transition diagram. These four models are tightly coupled, and it is assumed that they are consistent with one another based on a formal concordance process. Elements from each are transferred to the Colored Petri Net model. It is important that no additional information be incorporated in the CP net model that is not traceable to one or more of these four models. If it becomes necessary to make such additions or to make changes to the CP net model to make it execute properly, these changes must be also made in the static models and the C4ISR Architecture Framework products that are derived from them.

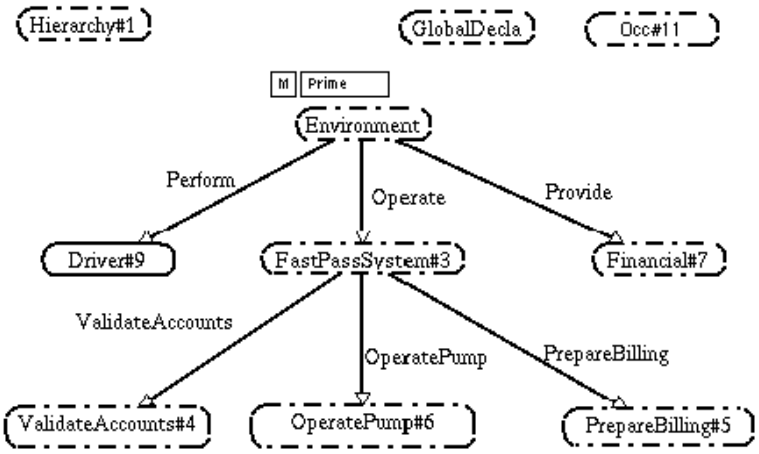


Figure 29. Hierarchy page of FastPass architecture CP net model.

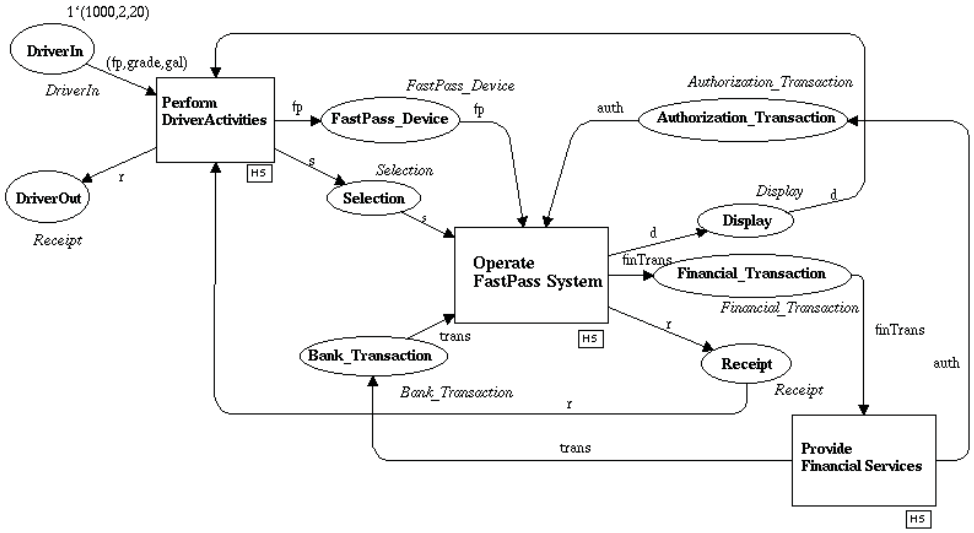


Figure 30. External system diagram representation in Design/CPN.

4.2.1. Implementation sources from IDEF0

One starts with the activity model. The CP net has the same hierarchical structure as the activity model. It has the same functional decomposition. The hierarchy page of the Design/CPN model is shown in Figure 29. Note its similarity to the functional decomposition shown in Figure 8.

The first page of the hierarchy is called the environment page and is similar to the External System Diagram of Figure 10. The CP net model of the environmental page is shown in Figure 30.

Each IDEF0 activity is converted into a transition; each IDEF0 arrow connecting two activity boxes is replaced by an arc-place-arc combination, and the label of the IDEF0 arc becomes the color set associated with the place. To illustrate this conversion, the IDEF0 model for the first level of decomposition is shown in Figure 31 and the corresponding CP net model page is shown in Figure 32.

Substitution Transitions are transitions that represent a subnet appearing in a child page in a Hierarchical CP net. They are used for each IDEF0 activity that is decomposed. The CP net page for the "Validate Accounts" activity is shown in Figure 33.

In order to execute the CP net model, it is necessary to provide the stimuli sources and receptor sinks for the external systems that interact with the system. The architect has several choices. As a first choice, the architect can play the role of each external system by manually providing and removing tokens that represent the inputs and response of the external systems as the simulation is run. For example, the architect would place a token in the input place that represents the Fast Pass Device and start the simulation. The simulation

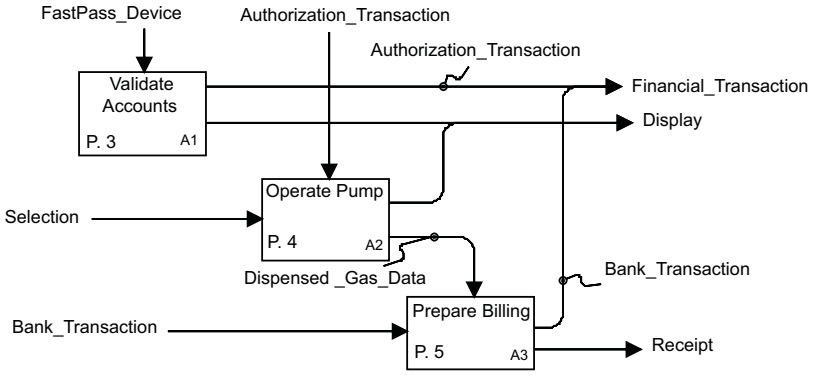


Figure 31. FastPass System IDEF0 model.

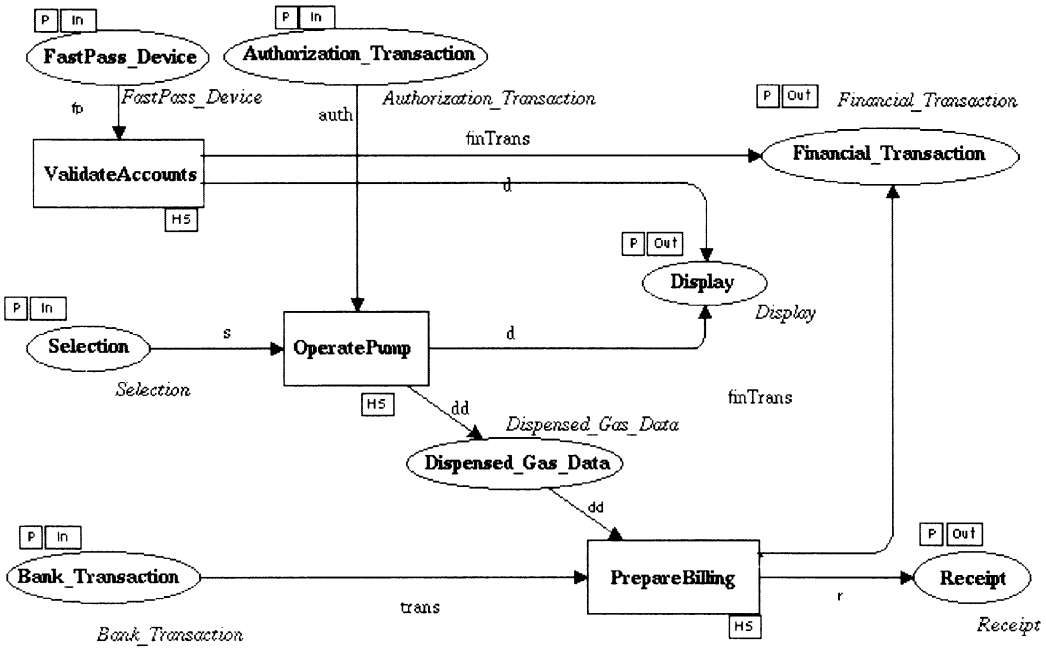


Figure 32. FastPass Color Petri net model.

would proceed to the point where the Authorization Transaction token was generated in the output place. The architect would remove this token and add the response of the Financial Institution in the Authorization Transaction input place and continue the simulation. It would run until it required a Selection input from the Driver. The architect would continue testing the execution of the model in this start and stop manner until the Driver received the receipt and the accounts were updated. An alternative is for the architect to create

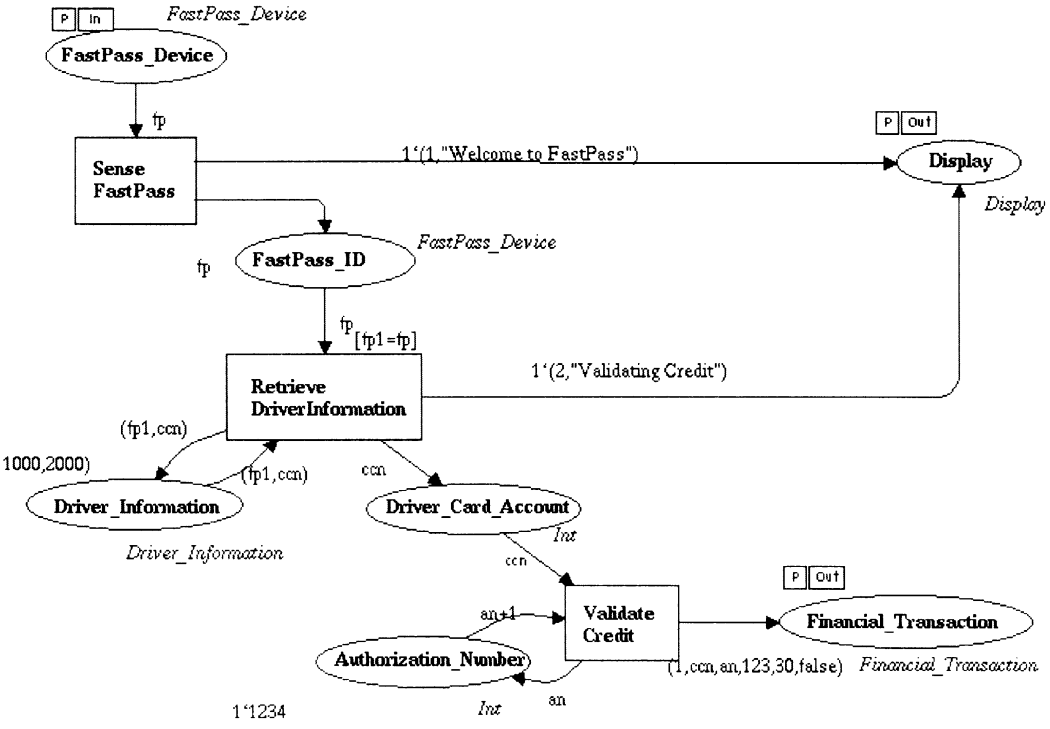


Figure 33. Color Petri Net FastPass System (Validate Accounts).

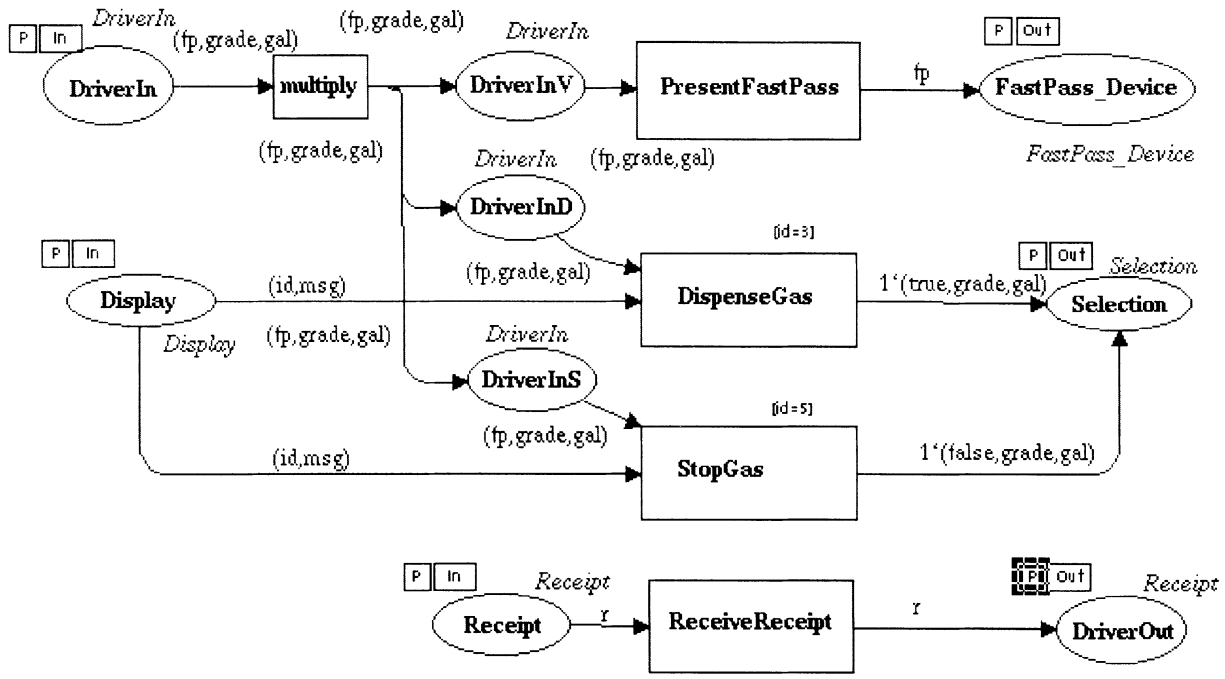


Figure 34. CP net model of the Driver external system.

additional CP net pages that model the actions of the external systems as they interact with the architecture model. These pages are connected to the appropriate input and output places of the architecture model. We have chosen this option in the example.

The CP net model page of the Driver is shown in Figure 34. The arrival of a Driver will be simulated by placing a token in the "Driver In" place. This token contains as attributes the FastPass device number plus the grade of gas and the number of gallons the Driver will pump. The transition named "Present FastPass" represents the Driver presenting the FastPass device to the pump. The Driver receives the various messages from the pump in the "Display" place. Some of the display messages are simply informative, i.e., "Welcome to FastPass," and others provide prompts to the Driver. We model the

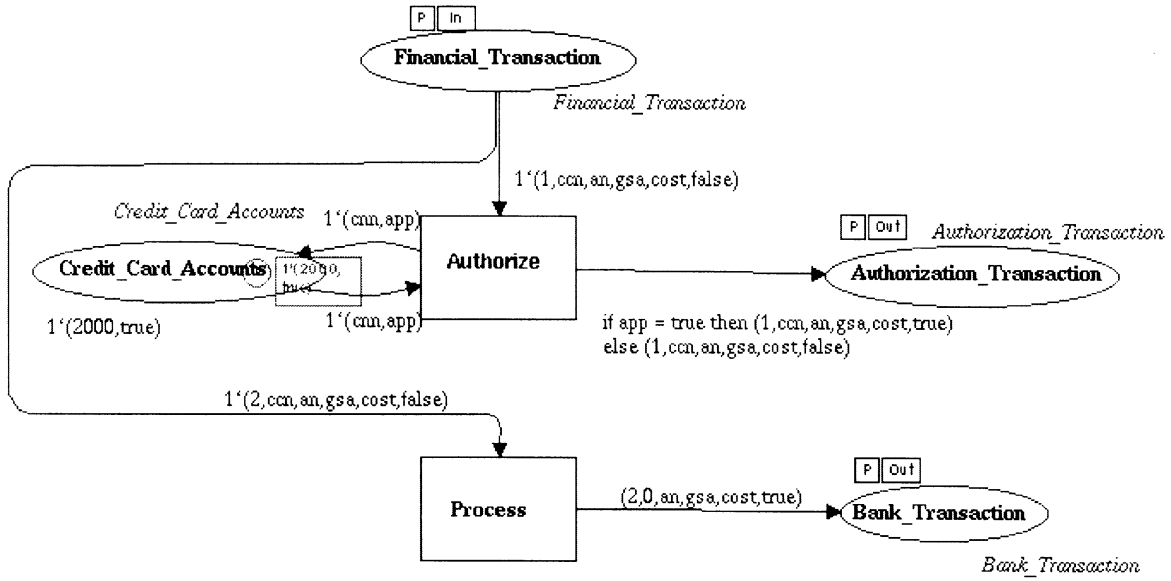
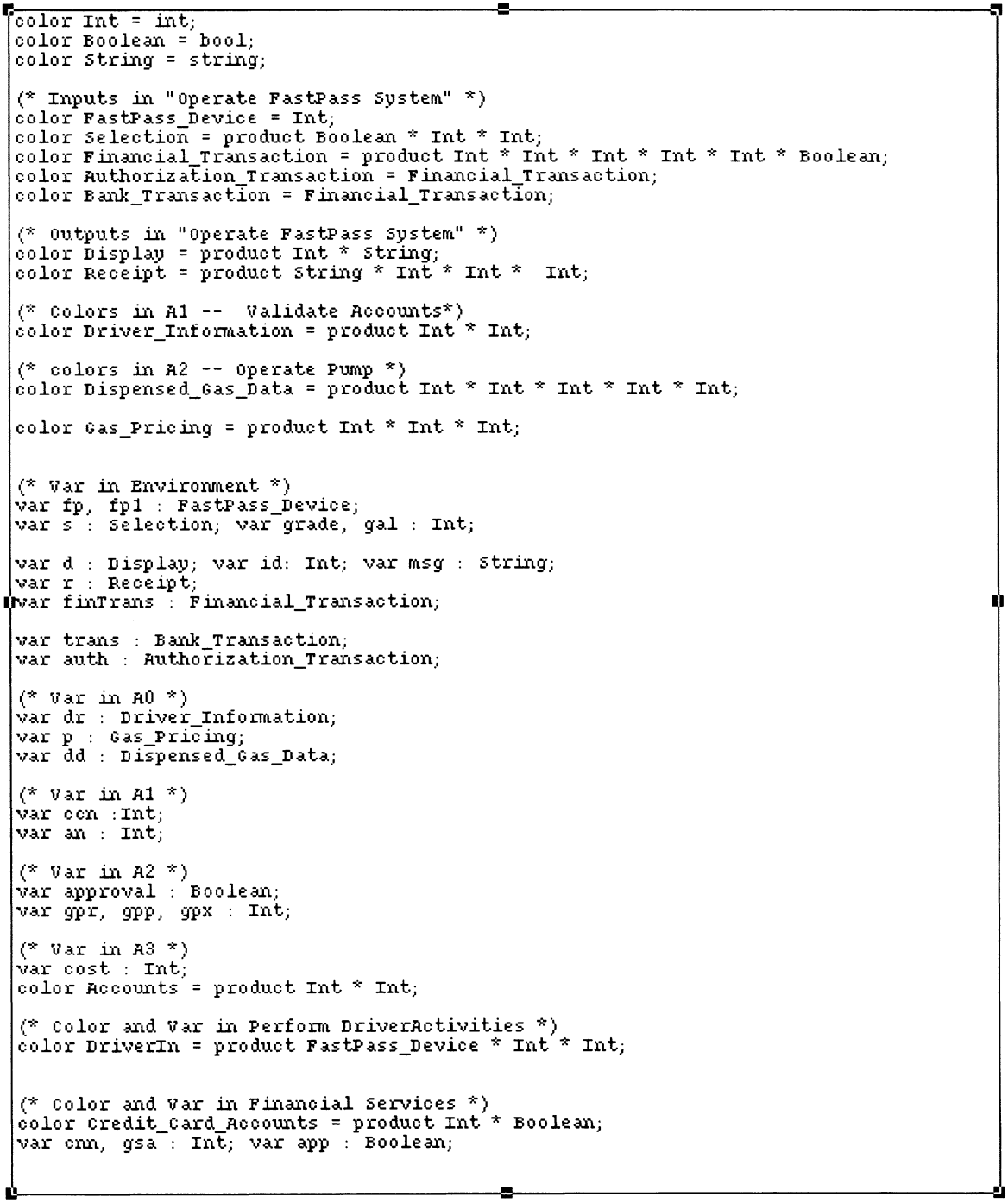


Figure 35. CP net model of the Financial Institution external system.

Driver's response to the prompts. Thus, there is a transition that models the Driver's response to the prompt to make a selection and a transition that represents the Driver stopping the pump when he is done. Finally, we model the Driver receiving the receipt.

The model of the Financial Institution is quite simple as shown in Figure 35. It contains two transitions. One represents the authorization process and the other the updating of the accounts process.

Care must be taken in constructing the models of the external systems. Our goal is not to model all of their actions but only the interactions they have with the architecture model. Making the external system models as simple as possible avoids having unnecessary com-



```
color Int = int;
color Boolean = bool;
color String = string;

(* Inputs in "Operate FastPass System" *)
color FastPass_Device = Int;
color Selection = product Boolean * Int * Int;
color Financial_Transaction = product Int * Int * Int * Int * Int * Boolean;
color Authorization_Transaction = Financial_Transaction;
color Bank_Transaction = Financial_Transaction;

(* Outputs in "Operate FastPass System" *)
color Display = product Int * String;
color Receipt = product String * Int * Int *  Int;

(* Colors in A1 --  Validate Accounts*)
color Driver_Information = product Int * Int;

(* colors in A2 -- Operate Pump *)
color Dispensed_Gas_Data = product Int * Int * Int * Int * Int;

color Gas_Pricing = product Int * Int * Int;


(* Var in Environment *)
var fp, fp1 : FastPass_Device;
var s : Selection; var grade, gal : Int;

var d : Display; var id: Int; var msg : String;
var r : Receipt;
var finTrans : Financial_Transaction;

var trans : Bank_Transaction;
var auth : Authorization_Transaction;

(* Var in A0 *)
var dr : Driver_Information;
var p : Gas_Pricing;
var dd : Dispensed_Gas_Data;

(* Var in A1 *)
var ccn :Int;
var an : Int;

(* Var in A2 *)
var approval : Boolean;
var gpr, gpp, gpx : Int;

(* Var in A3 *)
var cost : Int;
color Accounts = product Int * Int;

(* Color and Var in Perform DriverActivities *)
color DriverIn = product FastPass_Device * Int * Int;


(* Color and Var in Financial Services *)
color Credit_Card_Accounts = product Int * Boolean;
var cnn, gsa : Int; var app : Boolean;
```

Figure 36. Global Declaration Node.

plexity in the overall CP net model that can increase the analysis and evaluation effort unnecessarily.

4.2.2. Implementation Sources from IDEF1X

In the set of static architecture models, the IDEF1X data model describes the details of the input, controls, and outputs, of the IDEF0 activity model. The data model shows the composition of the data entities in terms of each entity's attributes and the relationships between them. The places in the CP net model will contain tokens that represent these entities. The type of tokens that each place can hold must be declared in the Global Declaration Node. Thus the IDEF1X entities are used to derive the names of color sets in the Global Declaration Node as shown in Figure 36.

We begin the construction of the color sets in the global declaration node by declaring atomic color sets Int (for integer), Boolean, and String. These will form the basis for the attributes that make up the color sets that will be used to specify the type of token each place can hold. Color sets that will be assigned to places are formed by using tuples created using the "product" constructor. Each color set that is assigned to a place has the same number and type of attributes as shown in the data model. In addition to the color set declaration, variables are associated with the color sets so they can be used in the arc inscriptions that implement the rules of the rule model.

4.2.3. Implementation Sources from the Rule Model

The information contained in the data model is used to specify the color sets and their respective domains, while the rules in the rule model result in arc inscriptions, guard functions, and code segments. In this example, all rules are expressed as arc inscriptions and guard functions.

4.2.4. Implementation Sources from the State Transition Diagram

The State Transition Diagram was created by following a key thread through the IDEF0 model. It reflects the initial conditions for the CP net model, the states the model should progress through from the initial state, and indicates dialogue between the system and the entities in the environment. It represents the behavior that the architect has created in the model and should be consistent with the behavior desired by the client. It will be used to verify that the model executes property.

4.3. Integration

The process of deriving the executable model invariably leads to some revision of the static models. Indeed, in creating the FastPass example, several minor changes had to be made in the models after undesired and unnecessary behavior was found in the executable model. It is most important that discipline be exercised so that any change introduced at the executable model level is reflected back in the static models. In this way, a documented and easily reviewed representation of the architecture can be maintained (traceability).

The executable model becomes the integrator of all the information; its ability to execute tests the logic of the model. The model can be executed to check its logical consistency, that is, to check whether the functions are executed in the appropriate sequence and that the data needed by each function are appropriately provided.

Since Colored Petri Nets with their dense annotation are not very easily understandable by the information system users, all the information gathered in the design and exploitation of the executable model needs to be brought back into the static models. This annotated and validated representation now constitutes a sound basis for system development.

5. LOGICAL AND BEHAVIORAL EVALUATION OF THE ARCHITECTURE

Once the executable model of the architecture has been created and debugged, it can be used to evaluate the architecture. Of course, the model can be run in the simulation mode to verify its logical and behavioral properties. But in addition to simulations, if Design/CPN[4] has been used, it is possible to quickly evaluate the behavior of the architecture using the built-in state space analysis tools. These tools have two main components, the occurrence graph analyzer and the state space report.

Design/CPN can generate the full occurrence graph for the FastPass architecture. It is shown in Figure 37. It is a directed graph with nodes and arcs. Each node is numbered and also shows the number of predecessor and the number of successor nodes (e.g., 1:2 means that the node has one predecessor and two successors). Each node represents a state of the CP net defined by the set of markings (tokens) of each place in the CP net. When Design/CPN creates the occurrence graph, it generates two types of information. First, it displays all of the tokens in all of the places in the model for each node in the occurrence graph. With each arc, it displays the name of the transition that fired causing the change of state and the bindings that enabled the transition in that

[4] Design/CPN is a software package developed and distributed by the University of Aarhus, Denmark. It has an Editor, a Simulator, and a set of analysis tools. See Kristensen, Christensen, and Jensen [1998].

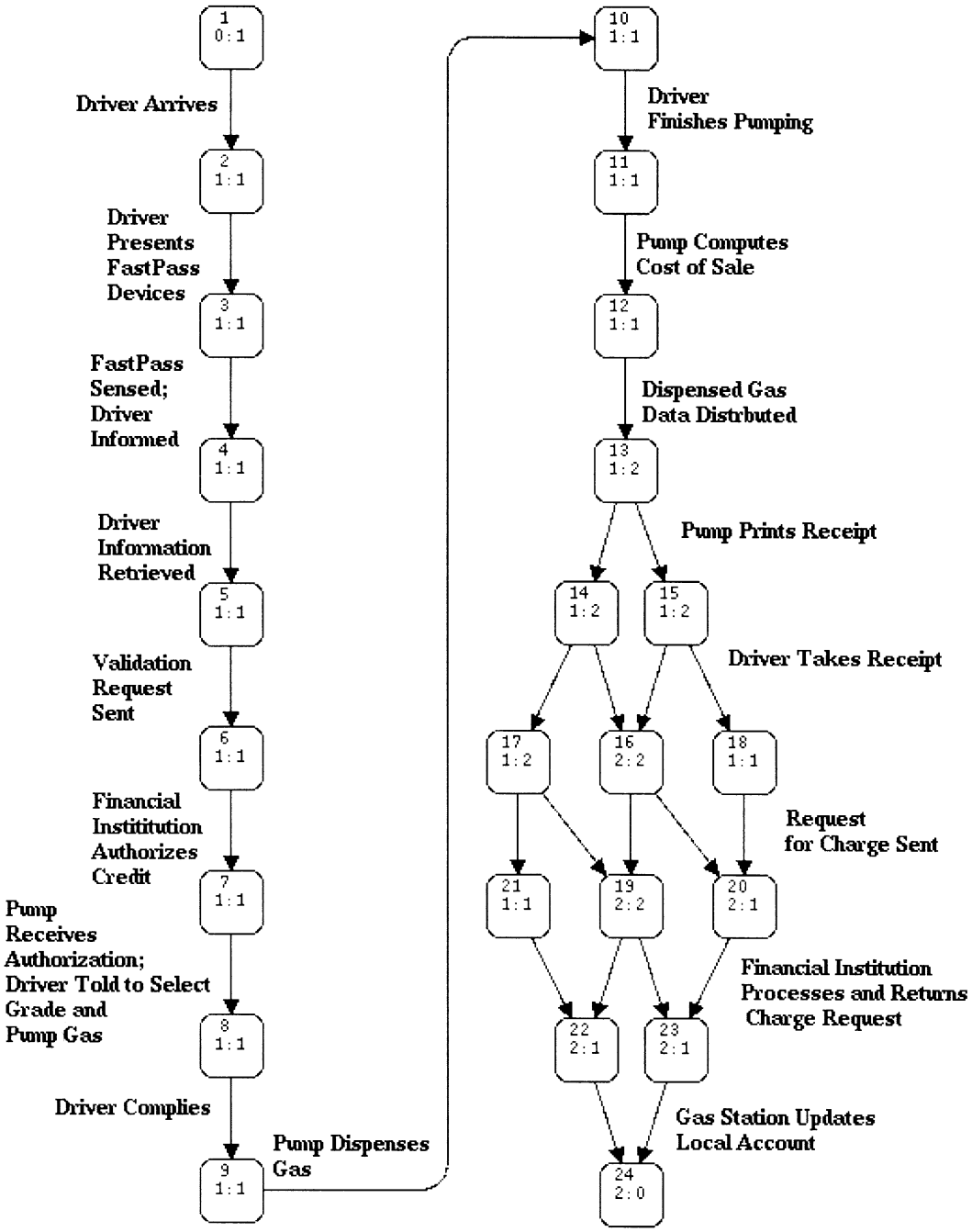


Figure 37. Occurrence graph of FastPass architecture.

firing. These displays can be shown or suppressed as desired by the user.

If the state space of the CP net is finite and small enough to be generated by the computer resources available, Design/CPN can generate all of the states that can be reached from an initial state by any allowable firing sequence. If the occurrence graph has one or more final states, then any path from the initial state to a final state represents a potential simulation run of the CP net. Thus the occurrence graph exhaustively shows all of the ways the model can execute from a given initial condition, i.e., it subsumes all the possible transition diagrams that start from the same initial state. Thus it can show all behavior for a given initial state. This description of behavior can be compared with the desired behaviors elicited from the clients during the domain

information gathering stage. Furthermore, new and potentially acceptable behaviors can be discovered and conveyed to the client.

A review of the occurrence graph of the FastPass architecture shows that it is consistent with the desired behavior. To aid in reading the graph, the detailed displays have been suppressed, and the transitions from state to state have been labeled with appropriate descriptors based on the firing information provided by Design/CPN. From the single initial state, the arrival of the driver, the execution sequences to State 13. The transitions correspond to those described in the State Transition Diagram of Figure 14 which represents the client's desired behavior. In State 13, the driver has finished pumping the gas and the pump has computed the final cost of sale. At this point, two process threads occur concurrently. The first thread is the printing of the receipt followed by the driver taking the receipt. The second thread is the pump sending the business transaction (charge request) to the financial institution, the financial institution updating its accounts and returning the business transaction, and the gas station updating its local account.

The fact that there are concurrent threads is reflected in the occurrence graph by the set of branches and joins starting from State 13. From this state there are nine paths through 11 states. Each path represents a feasible sequence of state changes that can occur as a result of the two concurrent threads. The nine paths represent all of the ways these two concurrent and independent threads can be interleaved. An important observation is that each path terminates in the same final state. This means that the two threads are independent, that is, one cannot effect the other. We know that the same receipt is printed and the accounts are updated in the same way regardless of the sequence of firings.

Every path through the occurrence graph is consistent with desired behavior. Because the occurrence graph contains every possible sequence of state changes from the initial state to the final state, it shows that the architecture design is sound and complete, and the state space analysis shows that the CP net is operating properly.

In addition to the occurrence graph, Design/CPN is capable of generating a state space report without executing the net. This report can be generated very quickly, even for large state spaces. It provides a great deal of useful information about the properties of the CP net that characterize the behavior of the net for a given marking. The report is divided into four sections: Statistical Information, Boundedness Properties, Home and Liveness Properties, and Fairness Properties. These reports can quickly show that the CP net model will behave in the desired manner. Useful information includes the number of final states, any transitions that will not fire from an initial marking, and the maximum number of tokens and their values that can appear in each place in the net. This can be basis for a formal analysis and evaluation of the properties of an information architecture.

6. PERFORMANCE EVALUATION

It is possible to extend the evaluation of the architecture from logical and behavioral evaluation to performance evaluation. To do this, the basic CP net model must be modified to reflect the use of resources and to incorporate processing times and transmission delays. In general, performance evaluation is a complex subject involving the determination of parameters that characterize the behavior and structure of system components (these are the performance parameters in SV-7), Measures of Performance (MOPs) that quantify attributes of system behavior, performance requirements, and Measures of Effectiveness (MOEs) that measure how well a system performs its function. The latter means that we must establish a method of comparing the measures of performance against the performance requirements to evaluate the measures of effectiveness.

One of the reasons for creating the executable model of the architecture is to use it to generate the data needed to determine measures of performance and measures of effectiveness for the architecture. The architect's clients are not only interested in the behavior of the architecture, but also are interested in how well the system's built-in conformance with the architecture meets certain performance goals or requirements. In general, creating a model that can answer performance questions requires that the model be based on the systems architecture view in which the functions of the operational view have been allocated to system components and elements, and a communications system has been specified. The performance parameters of these physical components of the architecture are then incorporated into the executable model. The model can then be instrumented to allow the collection of data that is needed to calculate the values of the measures of performance.

Because of its complexity, a detailed treatment of Performance Evaluation is beyond the scope of this article. However, we will illustrate some of the fundamental concepts of performance evaluation using our FastPass example.

In describing the FastPass system, we said that the main reason for the OilCo to install it was that it would attract more customers. This is because it would be more convenient to use than the conventional "pay at

the pump" systems that require the customer to get a credit card out of his or her wallet or purse. Implied in this rationale is that not only must the FastPass system have behavior that makes it easy to use, it must also operate in a timely fashion. It is this timeliness notion that is the main concept behind performance evaluation for FastPass. Indeed, it is the main measure of performance for our architecture. Furthermore, there is an implied requirement for this MOP: the time duration for the FastPass system to detect the FastPass device, retrieve the credit card information, and obtain credit approval must be at least as fast as the current "pay at the pump" systems. Let us assume that this requirement is less than 20 s, particularly if the system gives the Driver update messages as it goes through the approval process.

While the complete performance evaluation would require the inclusion of the system functions in the executable model, it is possible to obtain a high level quick look at the performance of the architecture using the CP net model already developed. This can be accomplished by changing the Design/CPN model from an untimed model to a timed one and by providing estimates of processing delays and the time required to transmit messages over the communications networks.

As described by Kristensen, Christensen, and Jensen [1998], Design/CPN supports time by using a global clock whose values represent *model time*. In addition to having values, tokens can carry a time value or *time stamp* designated by adding a suffix of the form @[t], where t is the value of the time stamp. The tokens that will carry time stamps must belong to a color set defined as *timed* in the Global Declaration Node. Intuitively, t is the model time which is the earliest time the token will be available for use: If the model time is less than the time stamp value t, the token can not enable any transition. If the model time is equal or larger than the time stamp value, then the token is available.

The time stamp of a token is updated in two ways: through the firing of a timed transition or because of a timed arc expression. A timed transition is a transition having a time region that contains an expression of the type @+(*expression*), where *expression* evaluates to a finite value that represents the duration of the process represented by the transition. When a timed transition fires, timed tokens are put in the output places as defined by the output arc expressions with a time stamp equal to the current model time augmented by the value of the expression of the time region of the transition. If a transition has no time region, the time stamp associated with the timed tokens put in the output places is equal to the current model time.

Output arc expressions can also be timed by adding an @+(*expression*) as a suffix to the standard arc expression. A timed arc expression can only be used for arcs that connect a transition to a place whose color set has been defined as timed in the Global Declaration Node. When a transition fires, timed tokens are put in the output places as defined by the output arc expressions with a time stamp equal to the current model time augmented by the value of the time expression of the arc expression.

This use of both time regions for transitions and timed arc inscriptions provides flexibility in the use of time in CP nets. Sometimes it is more natural to associate a time delay with a process that is modeled by a transition, and other times a time delay associated with an arc that represents the transfer of the output of a process to another process is more appropriate. Both models can be used at the same time in a model. When a timed arc expression is used with a timed transition, the time expression of the arc expression is added to the transition timed region so that the output token has a time stamp equal to the current model time augmented by the sum of the two time regions. We can use these two means of incorporating time in the FastPass model to characterize the performance of the architecture. We begin by estimating the processing time of each of the leaf functions of the functional decomposition and communications time delays for each of the arcs. These are summarized in Tables XV and XVI.

For this analysis, we assume that most of the processes associated with the gas station pump and office will take milliseconds compared to seconds to send messages over the wide area networks to retrieve Driver information, request credit authorizations, and update financial accounts. As a result, we set the time delay for all gas station activities to zero except for the printing receipt function. We estimate the time to access the Central database to retrieve the Driver information will take 10 s, and the time for the Financial Institution to perform the credit check function and update the account function will take 5 s each. The function to update the gas station accounts also will take 5 s. We also estimate that it will take 5 s to print the receipt.

We assume that the FastPass system relies on communications services that can be obtained from standard vendors. We estimate that the speed of service will be 3 s or less.

These processing and communications delays are incorporated into the CP net in three steps. A time region specifying the time duration is added to each transition with a nonzero processing duration. A time delay expression is added to each output arc from the producing functions of Table XVI that is equivalent to the estimated communications delay. Finally, the Color Sets of the Information Elements of Table XVI and the output places of the transitions with time regions are

Table XV. Fifteen Estimated Process Durations

Function/Transition	Estimated Duration	Output Place
Sense FastPass	0	Display
Retrieve Driver Information	10	Display, Driver Account
Validate Credit	0	Financial Transaction
Receive Authorization	0	Display, Authorization Transaction
Dispense Gas	0	Display, Dispensed Gas Data
Compute Cost of Sale	0	Dispensed Gas Data
Request Charge	0	Financial Transaction
Update Accounts	0	(none)
Print Receipt	5	Receipt
Financial Institution:Authorize	5	Authorization Transaction
Financial Institution:Process	5	Bank Transaction
Driver:Present FastPass	0	FastPass Device
Driver:Dispense Gas	0	Selection
Driver:Stop Gas	0	Selection
Driver:Receive Receipt	0	Driver Out

declared as timed. We also declare the Driver Out color set as timed so we can determine when the Driver departs. The CP net model will now execute and provide time stamps for the tokens in each of the timed places.

Note that we did not provide time durations for Driver actions or for the time it takes to pump the gas. This is because we are only interested in the time it takes to get the credit approval after the Driver presents the FastPass device and the time it takes for the drive to get his receipt after he/she stops pumping gas. Of course, these time durations also could be added to determine the total time a customer spends at a pump, but since they are the same whether FastPass is used or not, they can be suppressed for this calculation.

By running the simulation, we can collect data to calculate the time duration of various activities for the model. In particular, we find that it takes 21 s from the time that the Driver presents the FastPass device until the credit is approved. It takes only 5 s for the Driver to receive the receipt after he/she stops pumping the gas. The Gas station updates its records 11 s after the Driver stops pumping.

We note that time to authorize the pumping of gas exceeds the original requirement of 20 s by 1 s. At this point we have two choices. First we can try to see if the time can be improved by modifying the architecture. This may be done by changing the structure of the architecture or by changing the performance parameters of key processors or the communications links. If these changes are not possible or impractical, we can suggest that the requirement be adjusted to allow the architecture to meet the requirements. By revealing this type of performance characteristics, the executable model allows the architect to discuss behavioral and performance choices with the client.

One important characteristic of Colored Petri Nets is that the occurrence graph of a timed CP net is a subgraph of the same CP net that is untimed for a given initial marking. This means that every state in a timed CP net exists in the untimed net, i.e., if there were no undesirable states in the untimed CP net, there will be none in the timed CP net.

We can quickly generate the timed occurrence graph of the FastPass model and compare it with that of the untimed CP net. We have superimposed this occurrence graph on the untimed occurrence graph as shown in Figure 38. Again we have suppressed the detailed annotations of the states and the transitions and substituted more readable descriptions for the transitions. The time of the firing of each transition has also been provided.

Several observations are in order. From a structural point of view, the subgraph of the untimed occurrence graph shows that the branching and joining that was associated with the two concurrent threads has been reduced to a single branch and join after State 13. This branch occurs because both the "Request Change" and the "Print Receipt" transitions are concurrently enabled.

Table XVI. Estimated Communications Delays

Producing Function	Information Element (Color Set)	Receiving Function	Communication Delay
Validate Credit	Financial Transaction	Financial Institution: Authorize	3
Financial Institution: Authorize	Authorization Transaction	Receive Authorization	3
Request Charge	Financial Transaction	Financial Institution: Process	3
Financial Institution: Process	Bank Transaction	Update Accounts	3

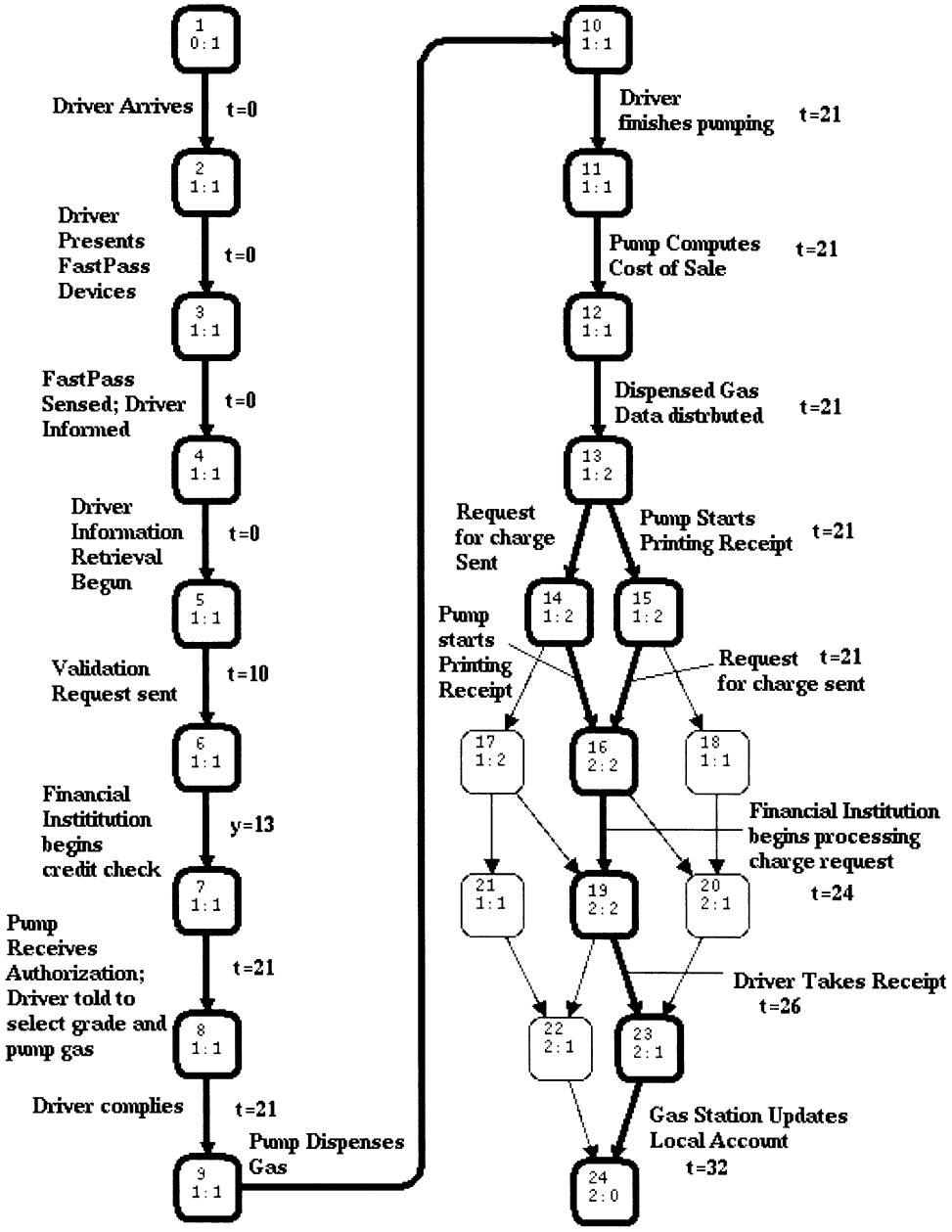


Figure 38. Occurrence graph of timed CP net model of FastPass architecture.

They can fire in either order. No other transition is ready to fire until after both of these transitions have occurred. Once they have both occurred, the time durations and delays cause the sequence of remaining state changes to be unambiguous, resulting in a single chain in the occurrence graph.

The second observation is more subtle. The annotations have been modified slightly from those used on the untimed occurrence graph. This was done for the firing of transitions that had time regions. For example, the firing of the print receipt transition on the untimed occurrence graph was simply labeled "Pump Prints Receipt." On the timed occurrence graph the annotation has been changed to "Pump Starts Printing Receipt." This subtle change is required because of the method used by Design/CPN to incorporate time in CP nets. When a timed transition fires, it fires instantly, causing a change of state. New tokens are created in the output places of the transition, but they are not immediately available due to their time stamps. Thus we interpret the firing of such transitions as meaning the start of the process represented by the transition. The result of that process will not be available until after the time duration associated with the transition. In the FastPass example, the

pump starts printing the receipt at time t =21 but the receipt will not be available until 5 s later at t = 26. The pump also generates the request for charge at the same time it starts the receipt printing. This request is generated instantly and is sent to the Financial Institution. Because we have modeled the communications delay of 3 s via the time expression on the arc, this request for charge is available at the Financial Institution at time t = 24. Thus the Financial Institution starts processing the request at time t = 24 before the driver can receive the receipt.

One of the advantages of using Petri Nets as the executable model is that they provided many analysis tools that can be used to evaluate the architecture. This simple examples illustrates how the occurrence graph can be a powerful way to visualize the behavior of the model. The state and transition information generated by the occurrence graph analyzer of Design/CPN provides most of the data needed for characterizing performance of the architecture. For example, the performance analysis could be expanded to estimate the timing of the various messages presented to the Driver. Each of these time delays represents a potential MOP that characterizes the performance of the system. Developing other analysis techniques for evaluating architectures using Petri Nets is an active area of research.

7. CONCLUSIONS

We have discussed the creation of information architectures in general and explored a process for creating the Essential and Supporting Products of the DoD C4ISR Architecture Framework Version 2.0. The Framework, by its depictions of example products, has a Structured Analysis bias. We have described the Structured Analysis approach, with roots in systems engineering, and how the architect can use the tools and techniques of Structured Analysis to produce a coherent set of products for the Operational and Systems views. The approach provides the necessary and sufficient set of information for creating executable models of the architecture that can reveal its logical, behavioral, and performance characteristics of the architecture.

The Framework describes a set of products or views of the architecture. It does not provide or recommend a process for creating these products. We have developed a strawman process based on Structured Analysis by reverse engineering from the set of products specified by the Framework. We have provided a simple example as an existence proof that the process can work. We recognize that each organization undertaking the development of an information architecture in the C4ISR domain may have its own preferred approach. We offer this approach that is based on the interrelationship of the products through the common data elements they contain as a means of reviewing any proposed approach to determine whether it preserves these relationships and whether it is capable of producing the particular set of products needed for the problem at hand.

Our conclusion is that it is feasible for the architect to use the Structured Analysis tools and techniques to create an information architecture. Once created, the C4ISR Framework products can be derived from the information contained in the Structured Analysis constructs. Of course, executable models can also be created and used as a focus of discourse with the customers of the architecture, even though these are not required by the Framework.

As was briefly discussed, there is a second approach to developing information system architectures that is a current area of research. The Object-Oriented approach has its roots in software systems engineering and may have several advantages over Structured Analysis for developing architectures of the types of information systems of interest to DoD. One main advantage is that newly trained engineers and computer scientists understand Object Orientation much better than Structured Analysis. The third paper of this three part series [Bienvenu, Shin, and Levis, 2000] explores Object Orientation for creating information system architectures in general as well as the prospects for converting the Object-Oriented products into the Essential and Supporting products of the C4ISR Architecture Framework.

APPENDIX

The architecture products defined in the C4ISR Architecture Framework document, Version 2, are listed below. For a description, see C4ISR [1997] or Levis and Wagenhals [2000].

Table IA. All View Products

Applicable Architecture View	Product Reference	Architecture Product	Essential or Supporting
All Views (Context)	AV-1	*Overview and Summary Information*	**Essential**
All Views (Terms)	AV-2	*Integrated Dictionary*	**Essential**

Table IIA. Operational Architecture View Products

Operational	OV-1	*High-level Operational Concept Graphic*	**Essential**
Operational	OV-2	*Operational Node Connectivity Description*	**Essential**
Operational	OV-3	*Operational information Exchange Matrix*	**Essential**
Operational	OV-4	*Command Relationships Chart*	Supporting
Operational	OV-5	*Activity Model*	Supporting
Operational	OV-6a	*Operational Rules Model*	Supporting
Operational	OV-6b	*Operational State Transition Description*	Supporting
Operational	OV-6c	*Operational Event/Trace Description*	Supporting
Operational	OV-7	*Logical Data Model*	Supporting

Table IIIA. System Architecture View Products

Systems	SV-1	*System Interface Description*	**Essential**
Systems	SV-2	*Systems Communications Description*	Supporting
Systems	SV-3	*Systems² Matrix*	Supporting
Systems	SV-4	*Systems Functionality Description*	Supporting
Systems	SV-5	*Operational Activity to System Function Traceability Matrix*	Supporting
Systems	SV-6	*System Information Exchange Matrix*	Supporting
Systems	SV-7	*System Performance Parameters Matrix*	Supporting
Systems	SV-8	*System Evolution Description*	Supporting
Systems	SV-9	*System Technology Forecast*	Supporting
Systems	SV-10a	*Systems Rules Model*	Supporting
Systems	SV- 10b	*Systems State Transition Description*	Supporting
Systems	SV -10c	*Systems Event/Trace Description*	Supporting
Systems	SV-11	*Physical Data Model*	Supporting

REFERENCES

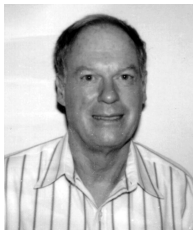
M.P. Bienvenu, I. Shin, and A.H. Levis, C4ISR architectures. III: An Object-Oriented approach for architecture design, Syst Eng 3, 288–312 (2000).

C4ISR Architecture Framework Version 2.0, C4ISR Architecture Working Group, Department of Defense, Washington, DC, December 18, 1997.

K. Jensen, Coloured Petri Nets, Springer-Verlag, Berlin, 1992.

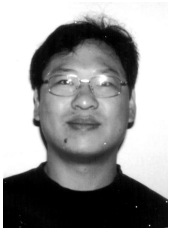
L.M. Kristensen, S. Christensen, and K. Jensen, The practitioner's guide to Coloured Petri Nets, Int J Software Tools Technol Transfer (1998).

A.H. Levis and L.W. Wagenhals, C4ISR architectures. I: Developing a process for C4ISR architecture design, Syst Eng 3, 225–247 (2000).



Lee W. Wagenhals is a Research Associate Professor with the Center of Excellence Command, Control, Communications, and Intelligence at George Mason University in Fairfax, Virginia. He is associated with the System Architectures Laboratory. His education includes the B.S. in electrical engineering from Lehigh University (1965), the M.S. from the Air Force Institute of Technology (1971), and the Ph.D. in Information Technology from George Mason University (2000). He has taught graduate and undergraduate System Engineering courses in modeling and analysis of dynamic systems at George Mason University (1996-98). Dr. Wagenhals has over 20 years of experience with the U.S. Air Force where he served on several assignments in area of both the operations and the research and development of C3 systems. He served for 5 years with the secretariat of the Air Force Scientific Advisory Board and participated in five summer studies on topics ranging from integrated avionics, to Global Reach Global Power. He joined George Mason University in 1992 where he has worked on a variety of research tasks related to C3 of Ballistic Missile Defense and the design and analysis of information systems architectures for the IRS. He was selected for the Army Summer Faculty Research Program in 1995 and performed research on digitization of the Battlefield for the Army Operational Test and Evaluation Command. Dr. Wagenhals' research interests include the design and evaluation of C3 and information systems architectures to distributed decision-making systems.

Insub Shin is a Ph.D. student in the doctoral program in Information Technology at George Mason University. He is currently conducting research at the System Architectures Laboratory in the Center of Excellence in Command, Control, Communications, and Intelligence. His education includes a B.S. in Mechanical Engineering from the Korea Military Academy (1985) and an M.S. in Telecommunications System Management from the U.S. Naval Postgraduate School (1990). He has over 10 years of experience with the Korean Army as a signal officer. He has taught information and communications technology in the Korean Army Signal School (1992–1994) and participated in the development of tactical C3 systems. His research interests include the design and evaluation of C3 information system architectures, modeling command and staff organizations for the information warfare era, and systems integration.



Daesik Kim is a Ph.D. student in the Information Technology and Engineering program at George Mason University. He is a graduate research assistant in the System Architectures Laboratory. His education includes a B.S. degree in Computer Science from Sogang University (1988) and an M.S. degree in Computer Science from Yonsei University (1990). He has been working as a senior researcher for the Agency for Defense Development in Korea since 1990. His areas of interest are Petri Nets, Discrete Event Systems, Distributed Computing, and Object-Oriented Computing.

Alexander H. Levis is University Professor of Electrical, Computer, and Systems Engineering. He is associated with the C3I Center, where he heads the System Architectures Laboratory. He served as Chair of the Systems Engineering Department in 1992–1994, and then again in 1996–1997. He was educated at MIT, where he received the B.S. (1965), M.S. (1965), M.E. (1967), and Sc.D. (1968) degrees in mechanical engineering with control systems as his area of specialization. He also attended Ripon College where he received the A.B. degree (1963) in mathematics and physics. He taught systems and control theory at the Polytechnic Institute of Brooklyn from 1968 to 1973 and, for the following 6 years, he was with Systems Control, Inc. of Palo Alto, California, where he was manager of the systems research department. From 1979 to 1990 he was a Senior Research Scientist at the MIT Laboratory for Information and Decision Systems. He joined George Mason University and the C3I Center in 1990. In 1992, he received the Distinguished Faculty Award. Dr. Levis is a Fellow of the Institute of Electrical and Electronic Engineers (IEEE) and past president of the IEEE Control Systems Society from which he received in 1987 the Distinguished Member Award. He is also a Fellow of the American Association for the Advancement of Science (AAAS), a member of AFCEA, and a senior member of AIAA. Dr. Levis' research interests for the last 15 years include architectures for command and control, organization theory and design, human decision making, and distributed intelligence systems. He has published over 130 papers and many book chapters and has coedited three volumes on The Science of Command and Control published by AFCEA.